Titre: Title:	Mirage : un nouvel outil de développement pour la réalité virtuelle sous X Windows
Auteur: Author:	Christophe Maurel
Date:	1997
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Maurel, C. (1997). Mirage : un nouvel outil de développement pour la réalité virtuelle sous X Windows [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/6911/

UNIVERSITÉ DE MONTRÉAL

MIRAGE : UN NOUVEL OUTIL DE DÉVELOPPEMENT POUR LA RÉALITÉ VIRTUELLE SOUS X WINDOWS

CHRISTOPHE MAUREL

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

DÉCEMBRE 1997

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

0-612-38695-3

Canada

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

MIRAGE : UN NOUVEL OUTIL DE DÉVELOPPEMENT POUR LA RÉALITÉ VIRTUELLE SOUS X WINDOWS

Présenté par: MAUREL, Christophe

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. LAVOIE Jean, M. Sc. A., président

M. GRANGER Louis, M. Sc., membre et directeur de recherche

M. MARCEAU Richard J., Ph. D., membre et codirecteur de recherche

M. MALOWANY Alfred, Ph. D., membre

REMERCIEMENTS

Je voudrais remercier tout d'abord Monsieur Richard J. Marceau pour m'avoir proposé ce projet et pour ses conseils précieux durant l'élaboration de ce travail de recherche et mémoire. Je tiens à remercier également Monsieur Louis Granger pour son aide concernant ce mémoire.

De nombreuses personnes ont contribué également à Mirage par des projets ou idées. Je remercie donc les étudiants Patrick Desbiens, Antoine D'Anjou, Frédéric Bélanger, Christian Auger et Elaine K. Tam pour leur aide.

Je remercie chaleureusement mes parents pour leur soutien.

Finalement, je remercie Le Fonds de Polytechnique pour son aide financière.

RÉSUMÉ

La réalité virtuelle (RV) a pris une place importante dans le milieu informatique depuis quelques années. Le projet ESOPE RV, un prototype de logiciel de formation d'opérateurs de réseaux en réalité virtuelle, a bien démontré son utilité. Un problème courant de ces applications est le coût élevé en matériel et logiciel requis pour développer de tels prototypes. Il est donc nettement avantageux de créer un logiciel de développement d'applications RV fonctionnant sur des stations PC courantes sous Linux. Nous proposons donc une nouvelle application nommée Mirage, qui résout ces problèmes. Developpé à partir de VR386, Mirage fonctionne sur toute station X Windows et il est disponible sans frais, tout en offrant une performance similaire à des outils commercialisés sur PC.

ABSTRACT

Virtual Reality (VR) has grown considerably in importance these past few years as computer graphics technologies have progressed greatly. ESOPE-RV, a prototype of an operator training system using virtual environment (VE) technologies , has proven its great usefulness. Problematic in these systems is the high cost of software and hardware required in their development. A VR application running on a standard PC using Linux does therefore prove extremely useful. A new application, called Mirage, proposes to solve the problems mentionned above. Developed from VR386, Mirage functions on numerous X Windows platforms, is freely available and offers similar performance to commercial applications running on a PC.

TABLE DES MATIERES

LISTE DES ANNEXES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES SIGLES ET ABBRÉVIATIONS

API	*Application Programmer Interface.* Bibliothèque de fonctions d'un logiciel servant à développer de nouvelles applications.
ASCII	*American Standard Code for Information Interchange.* Format informatique pour les caractères.
BSD	*Berkeley Software Distribution.* Distribution répandue de Unix
CAO	Conception Assistée par Ordinateur. Logiciels servant à dessiner et concevoir des plans, dessins, ou objets tridimensionnels.
CD-ROM	*Compact Disc - Read Only Memory.* Disque compact lu par laser capable d'enmagasiner environ 600 Mb d'informations
CPU	*Central Processing Unit.* Unité centrale de traitement d'un ordinateur
`DXF`	*Drawing Exchange File.* Extension des fichiers d'AutoCAD servant à l'échange avec d'autres logiciels.
`FIG`	Figure. Extension des fichiers utilisés dans Mirage et VR386 pour décrire une hiérarchie d'objets virtuels
GLUT	*Graphics Library Utility Toolkit.* Bibliothèque de fonctions servant à gérer les fenêtres et événements dans une application OpenGL.
GUI	*Graphical User Interface.* Interface usager d'un programme avec des menus.

HMD	*Head Mounted Display*. Casque de réalité virtuelle muni d'écrans et parfois d'écouteurs intégrés.
LCD	*Liquid Crystal Display*. Écran plat à cristaux liquides.
LOD	*Level Of Detail*. Niveau de détail associé aux représentations multiples d'objets virtuels.
Mesa	Bibliothèque graphique disponible gratuitement reproduisant 90% des fonctions d'OpenGL sur systèmes Unix et Windows.
Mb	*Megabyte*. Quantité d'information équivalent à environ 1 million d'octets.
MR Toolkit	*Minimal Reality Toolkit*. Logiciel de RV développé à l'Université d'Alberta utilisant la librairie OpenGL.
NFF	*Neutral File Format*. Extension des fichiers de WorldToolKit servant à l'échange avec d'autres logiciels.
OpenGL	*Open Graphics Language*. Bibliothèque graphique développée par Silicon Graphics pour l'infographie 3D.
PC	*Personal Computer*. Dans le texte, ordinateur de type 486, Pentium ou compatible.
PLG	*Polygon*. Extension des fichiers utilisés dans Mirage et VR386 pour décrire des objets virtuels constitués de polygones.
RV	Réalité Virtuelle.
SGI	Silicon Graphics Inc. Compagnie fabriquant des postes de travail

servant principalement en infographie de pointe.

Tcl — *Tool command language* (prononcé « tickle »). Langage ou script développé pour controller et étendre des applications existantes.

Tk — *Tool kit.* Script servant à créer des interfaces usagers (GUI) en combinaison avec Tcl.

VR — *Virtual Reality.*

VRML — *Virtual Reality Modelling Language.* Langage utilisé sur l'internet pour décrire des mondes virtuels.

WLD — *World.* Extension des fichiers utilisés dans Mirage et VR386 pour décrire des mondes virtuels.

WRL — *World.* Extension des fichiers de type VRML.

INTRODUCTION

La réalité virtuelle (RV) a pris une place importante dans le monde informatique depuis quelques années. Sa popularité a grandi, aidée en partie par les médias. Cette technologie a ses origines dans le milieu militaire car à ses débuts elle était très dispendieuse. On pense notamment aux simulateurs de vol de l'armée [Rolfe 1988]. Heureusement, les coûts en matériel de cette technologie ont chuté dramatiquement ces derniers temps la rendant plus accessible. Les applications de la RV se sont alors multipliées en industrie, en médecine [Bajura 1992], et en aérospatiale [Loftin 1994].

Cette technologie est très utile à l'enseignement en particulier. Des études démontrent que l'apprentissage par expériences personnelles est beaucoup plus efficace pour l'acquisition de connaissances que la lecture ou la visualisation d'expériences. A défaut de placer la personne dans l'environnement véritable qui peut être coûteux (ex : réparer un satellite sur une maquette grandeur nature qui soit une reproduction exacte), le milieu environnant peut être recréé avec des images de synthèse. Ce monde synthétique est appelé monde virtuel. Il peut comprendre tant l'aspect visuel que sonore de la manipulation. Si désiré, une rétroaction [*feed-back*] tactile est également possible [Mark 1996]. Pour l'instant, trois sens peuvent être stimulés en réalité virtuelle, soit la vue, l'ouïe et le toucher.

Dans cette introduction, nous ferons une revue des périphériques utilisés en réalité virtuelle, ainsi qu'un survol des logiciels de RV les plus répandus. Ensuite le sujet et le but de ce mémoire seront introduits, soit le logiciel Mirage.

Périphériques de visualisation

Les périphériques importants de la réalité virtuelle sont reliés à la visualisation ou à la navigation et à la manipulation d'objets dans le monde virtuel. Parmi les moyens de visualisation, on trouve le HMD (*Head Mounted Display*), les lunettes stéréoscopiques, le moniteur et certains environnements particuliers comme le tableau de bord d'un avion.

Le HMD est en général le moyen le plus coûteux de visualisation. Typiquement, il comporte deux écrans à cristaux liquides, soit un pour chaque oeil. A moins d'y mettre le prix, il offre une faible résolution graphique en comparaison avec les autres moyens mais donne une bonne impression de présence dans le monde virtuel. Ces casques sont souvent munis d'écouteurs donnant un son stéréo intégré. Les déplacements de la tête de l'utilisateur repérés par un système de repérage entraînent un changement correspondant de la scène vue à l'écran (*headtracking*). Il arrive qu'il y ait des délais entre le mouvement réel de la tête et le mouvement correspondant dans le monde virtuel. Ce délai (appelé aussi *lag* ou *latency*) peut produire des nausées chez les utilisateurs [Olano 1995]. Le casque HMD n'est pas pour le moment encore une solution au point.

Les lunettes stéréoscopiques sont un moyen moins coûteux de reproduire l'effet tridimensionnel de la vision binoculaire. Ces lunettes peuvent être polarisées, avec une

polarisation différente pour chaque oeil tel qu'utilisé dans les salles de cinéma IMAX 3D. Elles peuvent aussi fonctionner par obturation successive des images destinées à l'oeil droit et gauche [Hodges 1992]. Notons que cette obturation doit être synchronisée avec l'image sur l'écran de cinéma ou le moniteur. Cette synchronisation est effectuée généralement avec des rayons infrarouges (voir Figure 1).

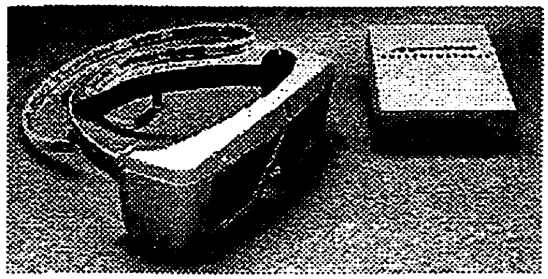


Figure 0-1 Lunettes stéréoscopiques et émetteur infra-rouge

Un simple moniteur peut suffire pour faire de la réalité virtuelle mais l'impression d'immersion dans un monde virtuel est faible avec ce moyen. Cette méthode de visualisation est appelée communément réalité virtuelle en aquarium [*Fish tank VR*].

Périphériques de navigation ou de manipulation

Ces périphériques permettent de se déplacer dans le monde virtuel ou de manipuler les objets virtuels. A la base, on trouve le clavier. Les touches fléchées du clavier permettent d'avancer, de reculer et de tourner sur soi-même dans le monde virtuel. La souris et la manette de jeu sont d'autres moyens pour naviguer dans l'environnement. Il existe également plusieurs types de souris 3D, par exemple la souris Cyberman de Logitech.

Pour la manipulation d'objets, le gant virtuel [Sturman 1994] est sans doute l'outil le plus approprié et le plus instinctif à utiliser. Le gant [*powerglove*] [Gradecki 94] (voir Figure 0-2) anciennement fabriqué par Mattel demeure populaire et abordable. Fabriqué à l'origine pour des consoles de jeux, il peut être connecté à un ordinateur personnel via le port série grâce à un circuit disponible gratuitement sur internet appelé *Menelli box* [`http://www.cms.dmu.ac.uk/~cph/menelli.html`].

La position spatiale du gant est repérée grâce à deux émetteurs d'ultrasons situés sur le gant. Le temps de propagation des ultrasons pour être mesurés par 3 récepteurs montés sur une barre en « L ». Ces temps varient en fonction de la position du gant, permettant ainsi de le localiser (*tracking*). La flexion des doigts est également mesurée. Sur chaque doigt du gant se trouve une encre conductrice dont la résistance varie en fonction de la flexion des doigts. Une boîte noire contenant un microprocesseur s'occupe de coordonner l'émission et la réception des ultrasons et de transmettre les données recueillies à l'interface.

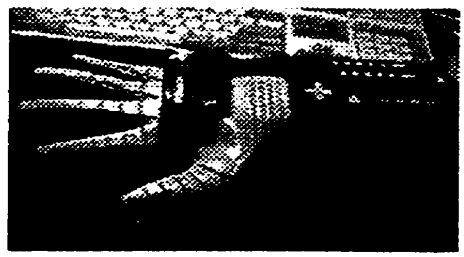


Figure 0-2 Gant Mattel Power Glove

OpenGL

Le langage graphique OpenGL [Segal 1994], [Neider 1993] fut développé par Silicon Graphics afin de faciliter la programmation d'environnements 3D. OpenGL est devenu un standard pour le développement de logiciels de réalité virtuelle d'abord sur les stations Silicon Graphics mais plus récemment sur d'autres plates-formes (i.e. Windows NT et 95). Parmi les applications écrites en OpenGL, on retrouve par exemple WorldToolkit [http://www.sense8.com/wtk.html] et la bibliothèque Open Inventor [Wernecke 1994], [Strauss 1992]. Originellement supporté par SGI, OpenGL est maintenant disponible sur plusieurs plates-formes Unix, entre autres AIX, Solaris, Linux, etc., ainsi que sur d'autres systèmes d'exploitation tels Windows 3.11 avec Win32s, Windows NT ou 95 et OS/2. Ce langage libère le développeur de nombreuses tâches ingrates de programmation de fonctions de base 3D. De plus, en réalisant certaines fonctions d'OpenGL directement sur les cartes graphiques (avec des processeurs spécialisés) au lieu de les émuler en logiciel, le processeur est libéré de nombreuses fonctions. La performance s'en trouve nettement améliorée.

MR Toolkit

La bibliothèque MR Toolkit fut développée à l'Université d'Alberta sur des postes Silicon Graphics. Le but de cette bibliothèque était de simplifier le développement d'applications en réalité virtuelle. MR Toolkit est écrit avec la bibliothèque OpenGL et se sert des capacités réseau des stations UNIX pour répartir les processus sur des plates-formes

multiples en réseau, accélérant ainsi la performance des applications. Ce logiciel est disponible gratuitement sur l'internet pour les institutions universitaires [Shaw, Green, Lian, Sun 1993].

WorldToolKit

WorldToolKit est un logiciel de la firme Sense 8, permettant de réaliser des applications en réalité virtuelle. Ce logiciel commercial est dispendieux mais demeure populaire car sa bibliothèque de fonctions est bien construite. Plusieurs plates-formes sont supportées (PC sous Windows et SGI entre autres) et de nombreux périphériques également. Le projet Esope-RV [Garant 1995], [Okapuu-Von Veh 1996] fut réalisé avec WorldToolKit sur des postes de travail R3000 et R4000 de SGI. Ce projet recrée l'environnement d'un poste d'Hydro-Québec en réalité virtuelle pour la formation des opérateurs de réseau. L'inconvénient de WorldToolKit est qu'une licence de plusieurs milliers de dollars est nécessaire pour chaque poste utilisant une de ces applications.

Rend386

Rend386 [Stampe, Roehl, Eagan 1993]; [Roehl 1994]; [Gradecki 1994] est le fruit d'une collaboration entre Bernie Roehl et Dave Stampe de l'Université de Waterloo. Ce logiciel, qui fut distribué gratuitement sur internet [`ftp://sunee.uwaterloo.ca/pub/rend386`], a permis à de nombreux intéressés de s'initier à la réalité virtuelle. Fonctionnant sur des ordinateurs bas de gamme de type 386, ce logiciel offre des bonnes performances car les auteurs ont tout misé sur la

rapidité de l'affichage. Une fois leur collaboration terminée, les auteurs ont réalisé chacun leur propre version améliorée de Rend386, soit Avril de Bernie Roehl, et VR386 de Dave Stampe.

Avril

Avril est conçu par Bernie Roehl de l'Université de Waterloo. Ce programme est sous la forme d'une bibliothèque de fonctions C qui peuvent être utilisées pour concevoir un logiciel de RV. Cette bibliothèque est uniquement disponible sur PC et comme le code ne contient aucune optimisation en assembleur, les performances sont moindres que VR386. L'avantage d'Avril est la bonne construction de la bibliothèque. Étant donné que le code source de celle-ci n'est pas disponible, il est impossible de « porter » ce logiciel à une autre plate-forme (à l'exception d'une initiative de l'auteur Bernie Roehl). Avril est un logiciel gratuit disponible sur l'internet:

[http://sunee.uwaterloo.ca/~broehl/avril.html].

VR386

VR386 est la création de Dave Stampe de l'Université de Waterloo. Ce logiciel est une amélioration de REND386, créé par Dave Stampe et Bernie Roehl. Une des améliorations principales a été la séparation du code assembleur et C, qui étaient souvent présents dans les mêmes modules, en des modules séparés.

VR386 est un logiciel gratuit, disponible sur l'internet par `ftp` anonyme [`ftp://psych.utoronto.ca/pub/vr-386`]. Le logiciel a été conçu pour le processeur 386 et ses descendants. Il fonctionne sur PC en mode VGA 320x200 et supporte plusieurs périphériques de RV tels que le gant de Mattel [*Powerglove*] et les lunettes LCD stéréoscopiques de la compagnie Sega. Il est utilisé par de nombreux amateurs de RV qui font ce qu'on appelle du *garage VR*.

Limitations de VR386

VR386 est un programme fonctionnant sous le système d'exploitation MS-DOS. Il est donc en principe limité à utiliser uniquement la mémoire conventionnelle disponible du PC, soit environ 620K au maximum. Par un moyen assez complexe, il est possible de se servir de la mémoire étendue du PC. VR386 exploite cette possibilité pour utiliser jusqu'à 4 mégaoctets de mémoire pour stocker les objets 3D. Notons que l'accès à cette mémoire est si complexe que son usage est déconseillé à tous sauf les programmeurs chevronnés de MS-DOS.

Donc, pour concevoir des applications avec VR386, il faut se limiter quant à l'utilisation de la mémoire conventionnelle. On s'aperçoit rapidement que la mémoire est très restreinte pour implanter des fonctions supplémentaires. Il serait donc utile de porter ce logiciel à UNIX pour éliminer ce défaut. Le code assembleur sera remplacé par du code en langage C. Cette nouvelle version a été baptisée Mirage [Tam, Maurel, Desbiens 1997] et fait l'objet principal de ce mémoire.

Pourquoi Mirage ?

Tous les logiciels mentionnés présentent des inconvénients. Les logiciels fonctionnant sous MS-DOS sont connus pour être difficiles à développer. Les logiciels commerciaux de RV sont généralement robustes et performants mais très dispendieux. Par exemple, l'utilisation de WorldToolKit requiert à la fois un coût d'achat de licence par poste de travail de plusieurs milliers de dollars (voir Tableau 1). Nous nous sommes donc proposés de créer un nouveau logiciel de RV à partir d'un logiciel gratuit disponible sur l'internet. La particularité de ce logiciel de départ doit être nécessairement la disponibilité entière du code source. Seuls VR386 et MR Toolkit satisfont ce critère. MR Toolkit étant très difficile à configurer et exécuter, VR386 est devenu le meilleur choix. Sa performance sur PC est très bonne. Son inconvénient majeur est qu'une partie du code est en assembleur. Il a donc fallu remplacer cette partie en C. Cela a donc mené à un nouveau logiciel de RV fonctionnant sous X Windows, baptisé Mirage.

Tableau 0.1 Comparaison de divers logiciels de RV

Logiciel	Prix de base	Licence	Disponibilité du code source	Mémoire limitée	Plate-forme
WorldToolKit	5000 $	4000 $	Non disponible	Non	SGI, Sun, PC
MR Toolkit	Gratuit	Gratuite	Disponible	Non	SGI
Avril	Gratuit	Gratuite	Non disponible	Oui	PC
VR386	Gratuit	Gratuite	Disponible	Oui	PC
Mirage	Gratuit	Gratuite	Disponible	Non	SGI, Sun, PC

Mirage possède plusieurs avantages sur VR386. L'avantage principal est que Mirage est entièrement codé en C, alors que VR386 est un mélange de C et d'assembleur. De plus, Mirage utilise la bibliothèque X Windows. Mirage peut donc supporter tous les postes de travail fonctionnant avec le logiciel d'exploitation UNIX avec X Windows. Cet avantage d'être multi-plateforme n'est pas une propriété de VR386 qui fonctionne uniquement sur des ordinateurs personnels ayant MS-DOS. Mirage supporte donc les postes de travail Sun, SGI, ainsi que Linux. Des modifications mineures seraient nécessaires pour supporter les stations IBM RISC et HP.

Mirage n'a pas les limitations de mémoire de VR386. VR386 était limité à 4 mégaoctets de mémoire. VR386 se limitait donc à environ 2000 polygones alors que Mirage peut aller jusqu'à 9000 polygones. (Notons que cette limite est modifiable; un monde virtuel représenté par 9000 polygones requiert un processeur très puissant pour être affiché à une vitesse raisonnable.)

UNIX offre des possibilités en matière de réseau qui sont inexistantes ou qui ne sont pas une partie inhérente de MS-DOS. La mise en réseau permet de communiquer aisément entre les processus exécutant sur le même poste ou sur un autre poste sur le réseau par l'intermédiaire des sockets. UNIX est également multi-processus, nous permettant d'exécuter des applications simultanément avec Mirage et de communiquer avec celles-ci en envoyant ou en recevant des informations.

Avec l'émergence de Linux [Welsh 1995], il est maintenant possible d'avoir des postes de travail UNIX performants à très faible coût. Linux coûte environ 30 $ sur cédérom. Il est également possible d'acquérir une version BSD. Le matériel nécessaire à Linux est le même que celui pour des ordinateurs fonctionnant avec les systèmes d'exploitation MS-DOS et Windows 3.1 ou 95. Pour le bon fonctionnement de Mirage, un Pentium cadencé 60 MHz avec 8 mégaoctets de mémoire vive est recommandé comme plate-forme minimale.

Mirage est donc un pas en avant pour la réalité virtuelle. Le code source C est entièrement disponible et on peut rajouter des fonctions ou le modifier si nécessaire. Mirage est multi-plateforme et économique (gratuit !). Il permet de faire de la réalité virtuelle sans investir dans un logiciel commercial coûteux.

Contributions originales de ce mémoire

- Remplacement des fonctions de rendu réaliste [*rendering*] écrites en assembleur de VR386 en langage C;
- Création d'un logiciel de réalité virtuelle fonctionnant sur X Windows;
- Création d'une interface Tcl/Tk pour Mirage;
- Intégration d'une main virtuelle pour simuler un gant muni de senseurs;
- Détection de collisions entre les objets virtuels.

Structure du mémoire

Le présent mémoire est donc structuré de la façon suivante : tout d'abord, la description fonctionnelle de Mirage est présentée, suivie de la description détaillée et de la conclusion.

Chapitre 1 - Description fonctionnelle de Mirage

L'utilité de Mirage a été discutée dans l'introduction. Dans le présent chapitre, on explique les caractéristiques principales d'un monde virtuel ainsi que l'interface de Mirage. Les menus sont décrits en détails ainsi que les touches de raccourci permettant un choix plus rapide des éléments des menus.

1.1 Un monde virtuel

Un monde virtuel peut être décrit comme une représentation artificielle d'un monde réel. Comme dans un monde réel, un monde virtuel peut contenir un certain nombre d'objets tels une chaise ou un bureau. Les objets du monde virtuel peuvent être animés et offrir la possibilité d'interagir entre eux. Alors que les objets du monde réel sont construits avec des atomes, les objets d'un monde virtuel se construisent avec des polygones. Un ensemble de polygones de couleurs différentes peut donner une approximation d'un arbre, d'une voiture ou d'une planète. La principale fonction de Mirage est donc dans la visualisation de mondes virtuels peuplés par des objets ayant certaines propriétés et comportements. Les objets eux-mêmes sont constitués d'un certain nombre de polygones.

1.2 Charger un monde virtuel

De nombreux mondes virtuels ont été créés pour Rend386 et VR386, les prédécesseurs de Mirage. Ces mondes sont disponibles gratuitement sur des sites internet (par exemple,

ftp://sunee.uwaterloo.ca/pub/rend386/worlds). Ces mondes virtuels sont décrits dans des fichiers dont l'extension est WLD. Les différents formats de fichiers supportés par Mirage sont illustrés à la Figure 1.2 et sont décrits plus loin dans ce document. Pour visualiser ces fichiers WLD, il suffit d'exécuter la commande:

```
mirage <nom du fichier wld>
ex : mirage chess.wld
```

Le monde sera chargé en mémoire et affiché à l'écran, comme le montre la Figure 1.1. Il suffit d'appuyer sur une touche du clavier pour se débarrasser du message de départ. On voit alors le monde virtuel, mais également notre position en haut à gauche, les axes x, y, z en haut à droite et le nombre d'images générées par seconde en bas à gauche.

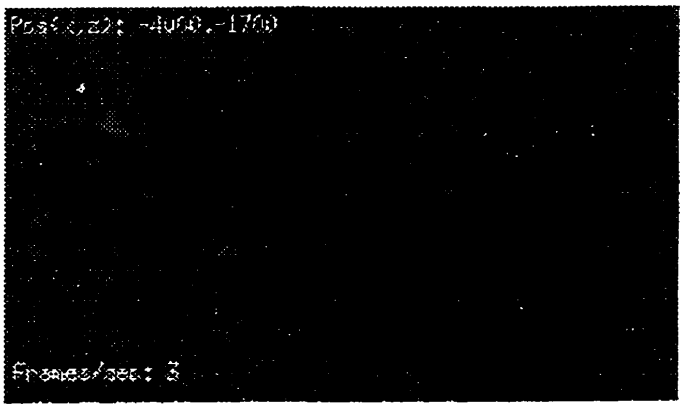


Figure 1.1 Un monde virtuel chargé dans Mirage

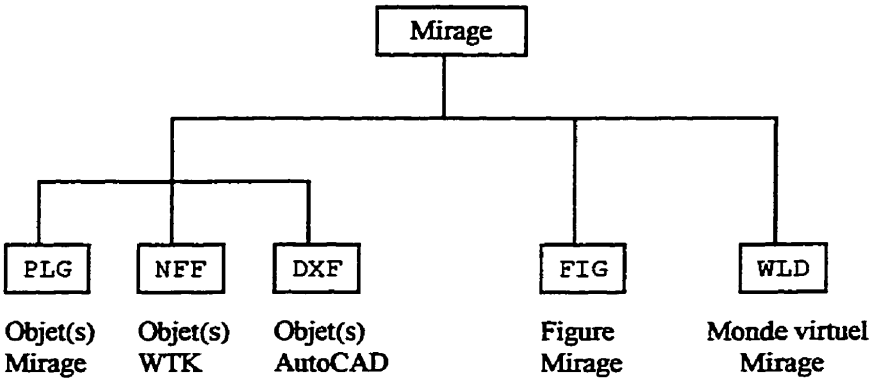


Figure 1.2 Formats de fichiers supportés par Mirage et leurs applications originelles

1.3 Navigation

1.3.1 Navigation avec le clavier

Les touches fléchées du clavier sont les touches de base pour se déplacer dans le monde virtuel. La figure suivante montre les quatre touches de base:

Tableau 1.1 Touches de navigation de base

Touche	Action correspondante
Flèche gauche	Tourne à gauche
Flèche droite	Tourne à droite
Flèche montante	Avance d'un pas
Flèche descendante	Recule d'un pas

D'autres touches permettent de monter, de descendre et d'accomplir d'autres actions. La liste est donnée dans le tableau suivant:

Tableau 1.2 Autres touches de navigation

Touche	Action Correspondante
Page Up	Monte le point de vue
Page Down	Descend le point de vue
+	Effet de loupe [*Zoom*] agrandir
-	Effet de loupe [*Zoom*] réduire
Home	Retour à la position de départ
U	Demi-tour (*U turn)*
R	Répète 100 fois le dernier mouvement avec les flèches

1.3.2 Navigation avec la souris

On peut également se déplacer en désignant une zone à l'intérieur de la fenêtre d'affichage avec le réticule. La fenêtre est divisée en quatre zones ayant chacune leur action correspondante. Par exemple, une action ou déclenchement d'une gachette [*click*] avec le premier bouton de la souris dans la zone 1 (voir Figure 1.3) fait avancer la caméra dans l'environnement virtuel.

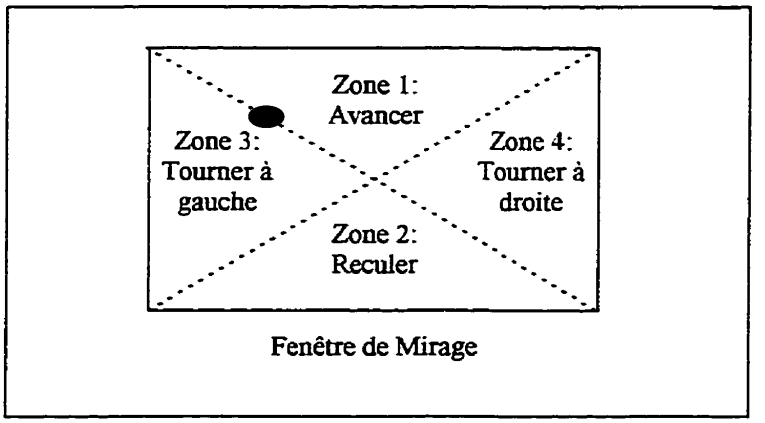


Figure 1.3 Les quatre zones de navigation pour la souris (Bouton 1)

Les autres boutons de la souris servent à d'autres actions. L'utilisation simultanée de deux boutons a également une action différente (voir Tableau 1.3). Un pivotement [*tilt*] est l'action d'incliner la caméra vers le haut ou vers le bas. Un balayage [*pan*] est une rotation de la caméra vers la gauche ou droite. Finalement, un roulement [*roll*] est une rotation de la caméra sur son axe horizontal.

Tableau 1.3 Navigation avec la souris

Bouton(s) appuyé(s)	Haut (zone 1)	Bas (zone 2)	Gauche (zone 3)	Droite (zone 4)
Bouton 1	Avancer d'un pas	Reculer d'un pas	Balayage à gauche	Balayage à droite
Bouton 2	Pivotement vers le bas	Pivotement vers le haut	Balayage à gauche	Balayage à droite
Bouton 3	Monter	Descendre	Pas vers la gauche	Pas à droite
Boutons 1 et 2	Effet de loupe grossissant	Effet de loupe rapetissant	Roulement à gauche	Roulement à droite

1.4 Menus de Mirage

Mirage comporte de nombreux menus accessibles avec le clavier dans la fenêtre principale de Mirage ou dans la fenêtre séparée de Mirage vue dans la Figure 1.4. Les commandes du menu permettent principalement de modifier le monde virtuel, de changer et d'afficher ses propriétés.

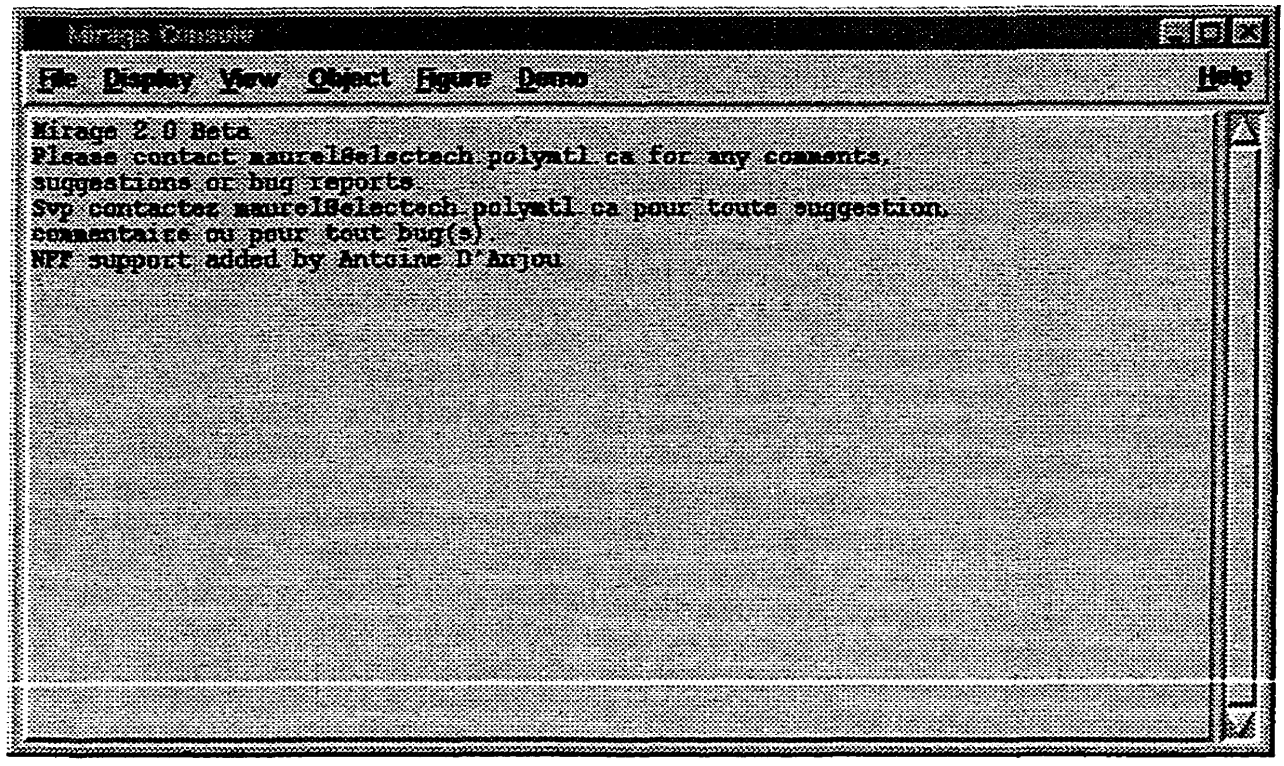


Figure 1.4 Console Tk de Mirage

1.4.1 Touches importantes

Les touches de raccourci données peuvent être activées soit en majuscule ou minuscule.

Quitter (« Q »):

Termine l'exécution du programme. Appuyez sur « O » pour « oui » et vous terminez l'exécution de Mirage.

Mode fil-de-fer [wireframe] (« W »):

Les polygones peuvent être affichés soit pleins ou en fil-de-fer. Cette touche permet de changer de mode.

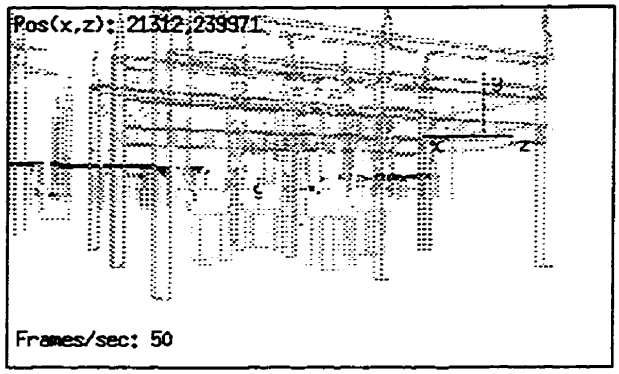


Figure 1.5 Mode fil-de-fer

Information (« I »):

Cette touche affiche des informations sur le monde virtuel, soit le nombre d'objets, de polygones et la position de la caméra.

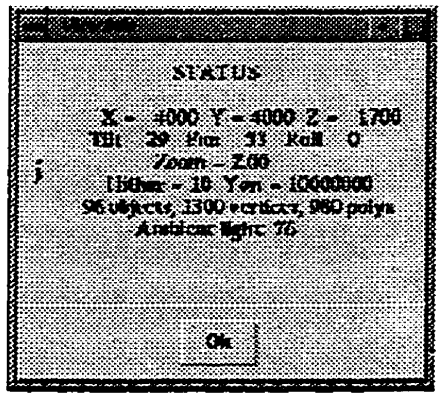


Figure 1.6 Image donnée par la commande Informations

Animation (« A »):

Les animations peuvent être désactivées avec cette commande. Avec les animations désactivées, on peut se déplacer plus rapidement dans le monde virtuel.

Aide (« H »):

Cette touche affiche la liste des touches de Mirage.

Navigation/sélection avec la souris (« J »):

La souris est au départ en mode navigation. En tapant sur la touche « J », la souris passe en mode sélection. On peut donc désigner un objet et il apparaîtra en évidence [*highlight*]. En appuyant à nouveau sur « J », on revient au mode navigation.

Affichage de la palette de couleurs (« C »):

Cette touche affiche la palette de couleurs de Mirage, telle qu'illustrée dans la figure suivante:

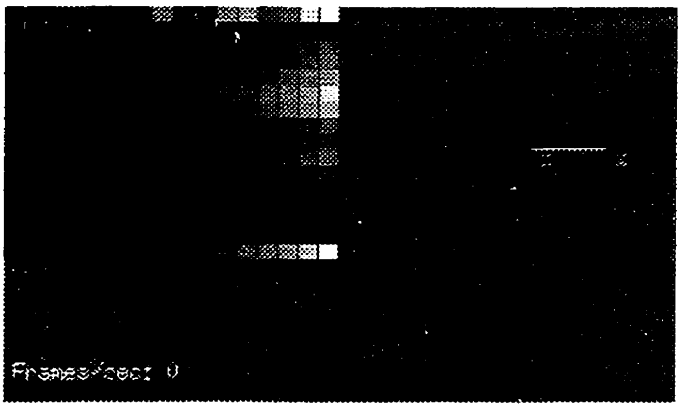


Figure 1.7 Palette de couleurs de Mirage

1.4.2 Menu Affichage

Ce menu contient diverses options pour l'affichage. Une option du menu peut être activée ou désactivée soit en désignant avec la souris une boîte de choix, soit en utilisant une touche de raccourci. Les options du menu précédées d'une boîte de sélection indentée sont à bascule.

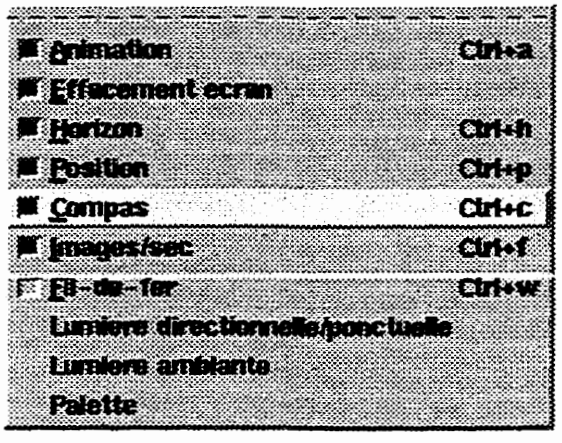


Figure 1.8 Commandes disponibles dans le menu Affichage

Désactiver/activer les animations (« A »):

Cette touche a le même effet que si on appuie directement sur « A ».

Désactiver/activer l'effacement de l'écran (« S »):

Par défaut, l'écran est effacé avant de dessiner les objets. On peut désactiver cet effacement si désiré.

Désactiver/activer l'horizon (« H »):

Par défaut, un horizon est dessiné pour mieux s'orienter. On peut le désactiver si désiré.

Changer la lumière ambiante (« A »):

On peut modifier le niveau de la lumière ambiante, donnant des objets plus sombres ou plus éclairés. Le niveau varie de 0 à 255. Par défaut, le niveau de lumière ambiante est mis à 65.

Changer entre lumière directionnelle et ponctuelle (« D »):

La lumière peut provenir d'une source ponctuelle ou d'une source à l'infini dont les rayons sont parallèles et directionnels.

Désactiver/activer l'affichage de la position (« P »):

Par défaut, la position de la caméra est affichée en tout temps en haut à gauche de la fenêtre. L'affichage de cette position peut être occulté si désiré.

Désactiver/activer l'affichage des axes ou compas (« C »):

Par défaut, les axes sont affichés à l'écran. Les axes peuvent être enlevés avec cette touche.

Désactiver/activer l'affichage du nombre d'images/seconde (« F »):

Par défaut, le nombre d'images par seconde est affiché en bas à gauche de la fenêtre. On désactive cet affichage avec cette touche.

Afficher la palette de couleurs de Mirage

La palette de 256 couleurs de Mirage est affichée. Chaque couleur est accompagnée de son numéro d'index.

1.4.3 Menu Vue

Ce menu contient diverses options qui peuvent être configurées. Ces options ont principalement un effet sur la caméra qui nous donne une vue sur le monde virtuel.

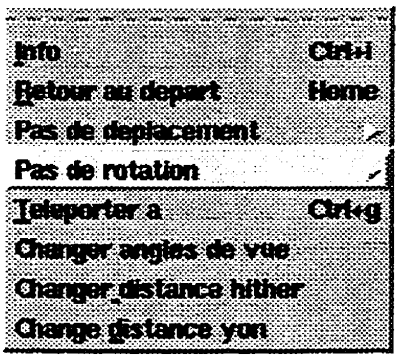


Figure 1.9 Commandes disponibles dans le menu Vue

Afficher de l'information sur le monde virtuel (« I »):

Equivalent à appuyer sur la touche « I » directement. Donne des informations sur le monde virtuel, soit le nombre d'objets, de polygones, etc.

Retour à la position de départ (« R »):

Cette touche est équivalente à la touche *Home*. La caméra ou position de l'observateur revient à la position de départ pour visualiser le monde virtuel.

Modifier le pas de déplacement (« M »):

Cette touche sert à modifier le pas de déplacement dans la scène si l'on désire déplacer la caméra plus ou moins rapidement.

Modifier le pas de rotation (« T »):

Cette touche sert à modifier le pas de rotation si l'on désire tourner plus vite ou moins vite.

Modifier l'angle de vue (« A »):

Cette touche sert à changer les angles de vue de la caméra. Trois angles doivent être donnés, soit les angles de pivotement, de balayage et de roulement. Ces angles sont relatifs aux axes x, y et z respectivement.

Modifier la position du plan avant du parallélépipède de vision (« H »):

La position du plan avant du parallélépipède de vision des objets est changée avec cette touche. Cela signifie que les objets entre le plan avant et la position de la caméra ne sont pas visibles à l'écran et sont donc éliminés lors des calculs d'affichage. L'effet de ces plans est expliqué au chapitre 3 dans le processus de rendu réaliste [*rendering*].

Modifier la position du plan arrière du parallélépipède de vision (« Y »):

Similairement, les objets plus lointains que ce plan deviendront invisibles.

1.4.4 Menu Objet

Ce menu permet de charger, de sauvegarder ou d'effacer des objets de Mirage.

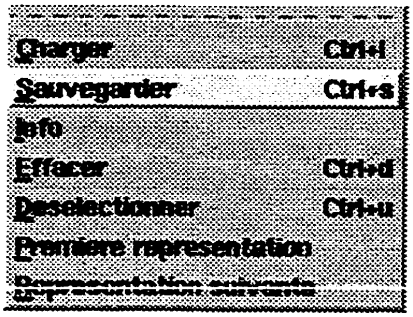


Figure 1.10 Commandes disponibles dans le menu Objet

Charger un objet (« L »):

Les objets décrits en format `PLG`, `NFF` ou `DXF` sont chargés avec cette commande. Les détails de ces formats sont décrits à la section 1.5. Il faut entrer le nom du fichier, et ensuite les facteurs d'échelle en x, y et z de l'objet. L'objet est inséré devant la caméra, à l'échelle appropriée. Si aucun facteur n'est spécifié (appui sur *Enter*), l'objet sera chargé avec les mêmes dimensions que dans le fichier (facteur d'échelle de 1). Il peut être déplacé par la suite à l'aide de la main virtuelle qui sera discutée à la section 1.4.7.

Sauvegarder les objets sélectionnés (« S »):

Les objets sélectionnés sont enregistrés dans un fichier de type PLG. Il faut donner le nom du fichier. Si plusieurs objets ont été sélectionnés, ils sont tous enregistrés dans le même fichier.

Afficher des informations sur les objets sélectionnés (« I »):

La position et l'orientation des objets est affichée, ainsi que leur nombre de polygones, etc..

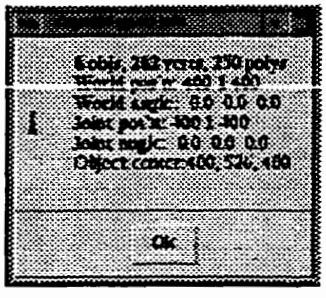


Figure 1.11 Affichage d'informations concernant un objet sélectionné

Effacer les objets sélectionnés (« D »):

Les objets sélectionnés sont éliminés du monde virtuel. Ils ne sont pas enlevés de la mémoire mais uniquement enlevés de la liste des objets du monde virtuel à afficher.

Retrait de la sélection des objets (« U »):

Tous les objets seront retirés de la sélection.

Changer à la première représentation de l'objet. (« F »):

Si l'objet a des représentations multiples, la première représentation sera affichée. La représentation multiple est expliquée à la section 1.5.4.

Changer de représentation (« N »):

La prochaine représentation de l'objet est affichée, si l'objet en possède plusieurs.

1.4.5 Menu Figure

Ce menu permet de charger et de manipuler les figures. Les figures sont des hiérarchies d'objets de type `PLG` (voir section 1.6).

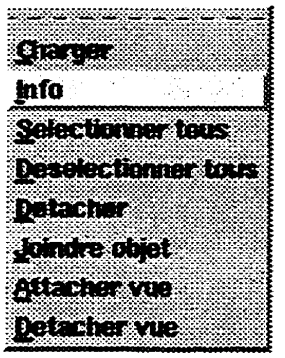


Figure 1.12 Commandes disponibles dans le menu Figure

Charger une figure d'un fichier `FIG`

Cette commande charge une figure à partir d'un fichier de type `FIG` (voir page 42). Une figure est une hiérarchie d'objets. Ce format de fichier est expliqué à la section 1.6.

Afficher de l'information sur la figure

Cette commande affiche la position, le nombre de polygones ainsi que d'autres informations d'une figure sélectionnée.

Sélectionner toutes les figures

Toutes les figures sont sélectionnées.

Enlever la sélection pour toutes les figures

Cette commande a pour effet de retirer toutes les figures de la sélection.

Détruire un lien avec un objet

Cette commande détache l'objet sélectionné d'une figure à laquelle il était rattaché.

Joindre un objet à une figure

Cette commande attache l'objet sélectionné à la figure.

Attacher le point de vue à un objet sélectionné.

Un lien est créé entre le point de vue et l'objet sélectionné. Chaque mouvement de l'objet engendre un mouvement correspondant du point de vue. Si l'objet est stationnaire, le point de vue reste figé au même endroit jusqu'à la destruction du lien avec l'objet.

Détacher le point de vue

Cette commande est l'opposée de la précédente. Elle détruit le lien entre le point de vue et l'objet.

1.4.6 Menu Démo

Il est possible d'enregistrer les déplacements de la caméra pour les reproduire par la suite. En activant l'enregistrement des déplacements (`Enregistrement`), ces derniers sont enregistrés dans un fichier jusqu'à ce que la commande `Arret` soit sélectionnée. Les déplacements enregistrés dans le fichier sont par la suite reproduits à l'aide de la commande `Jouer`. L'enregistrement tient uniquement compte des touches fléchées pour l'enregistrement des déplacements.

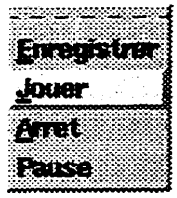


Figure 1.13 Commandes disponibles dans le menu Demo

1.4.7 Manipulation d'objets avec la main virtuelle

En l'absence d'un gant équipé de senseurs, Mirage possède une main virtuelle pour la manipulation des objets virtuels. Cette main a été intégrée à Mirage suite au projet de fin

d'études de Christian Auger [Auger 1996] (voir Figure 1.14). Les objets saisis par la main peuvent être déplacés ou tournés dans tous les sens. La main virtuelle est contrôlée à l'aide de la souris. Il est envisageable de remplacer ce contrôle par celui d'un gant équipé de senseurs. Les mouvements du gant seraient reproduits par la main virtuelle à l'écran.

Un mode anti-collision a aussi été créé. Dans ce mode, l'objet attrapé par la main est testé et bloqué s'il entre en collision avec une série d'objets sélectionnés. Les objets dans Mirage possèdent maintenant un parallélépipède englobant [*bounding box*] soit la boite minimale englobant l'objet. Cette propriété n'existait pas dans VR386.

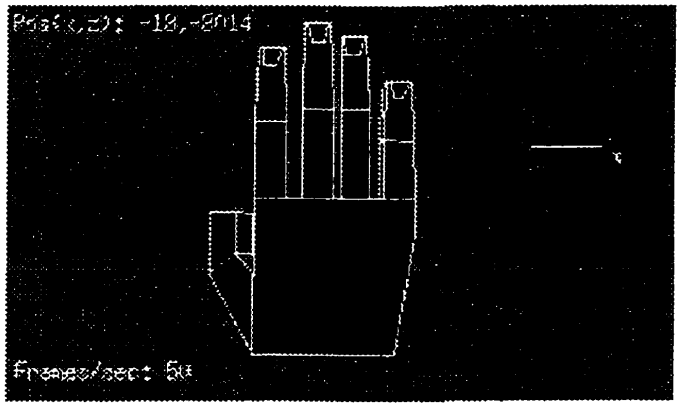


Figure 1.14 Main virtuelle manipulant un objet

La main virtuelle est activée par l'appui consécutif sur les touches « X » et « G ». La souris contrôle ensuite les déplacements de la main. Les objets peuvent alors être saisis en

les sélectionnant avec la souris et en appuyant sur la touche « A ». Avant et après la commande sélectionner, il faut appuyer sur la touche « J » pour changer de mode. Pour sortir de ce mode, il suffit d'appuyer sur la touche « G ».

Pour passer en mode anticollision, il faut tout d'abord se mettre en mode saisie [*grabbing*] et sélectionner tous les objets à considérer lors des tests d'anticollision. Le mode anticollision est activé en appuyant sur la touche « K ». Les objets saisis par la suite avec la main virtuelle seront « bloqués » lors de collisions avec les objets considérés lors du test d'anticollision.

1.5 Les objets dans Mirage

1.5.1 Les objets

Un objet dans Mirage est un ensemble de polygones. L'ensemble de polygones peut former un arbre, une sphère, etc... Un objet est donc défini par une liste de coordonnées de points dans un espace tridimensionnel et une série de polygones dont les points se trouvent nécessairement dans la liste précédente.

Il existe différentes façons de décrire les listes de points et de polygones représentant un objet et donc différents formats de fichiers définissant des objets. En effet, presque chaque logiciel possède son propre format de fichiers. Parmi les plus courants, on trouve le format `DXF` du logiciel de dessin AutoCAD. Mirage utilise le format `PLG`.

1.5.2 Les objets au format PLG

Les objets de Mirage sont généralement décrits sous la forme de fichiers de type PLG. Ce format est un format particulier utilisé par VR386. Dans ces fichiers, on retrouve une liste de coordonnées 3D spécifiant les sommets des polygones, et une liste de numéros de sommets constituant le polygone et ordonné dans le sens horaire. Chaque liste de numéro de sommets est précédée par un code couleur en hexadécimal et le nombre de sommets du polygone. Les polygones ne sont visibles que d'un seul côté. Il est donc important de placer les points du polygone dans le sens horaire, selon la face visible du polygone (voir Figure 1.15).

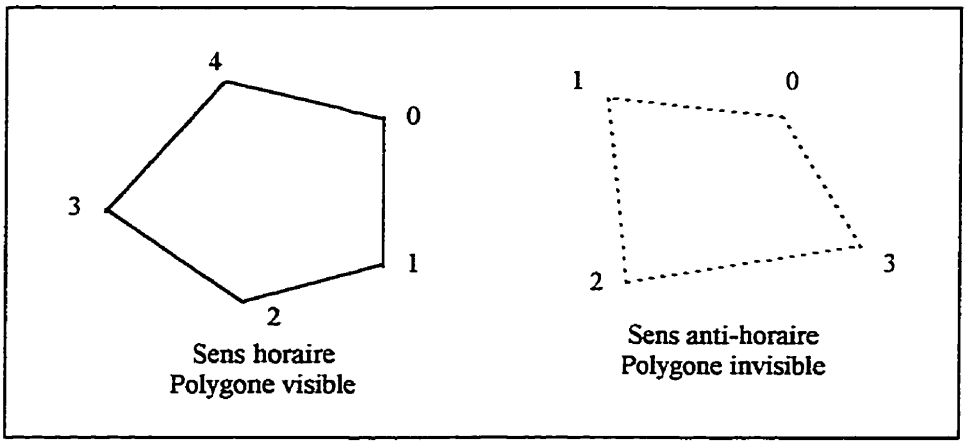


Figure 1.15 Ordre des points dans la définition d'un polygone

De plus, les polygones doivent nécessairement être convexes. Cette particularité permet d'optimiser l'affichage des polygones à l'écran. Les polygones concaves donnent donc

des résultats imprévisibles. Notons de plus que dans Mirage, un polygone peut au plus être composé de 20 points.

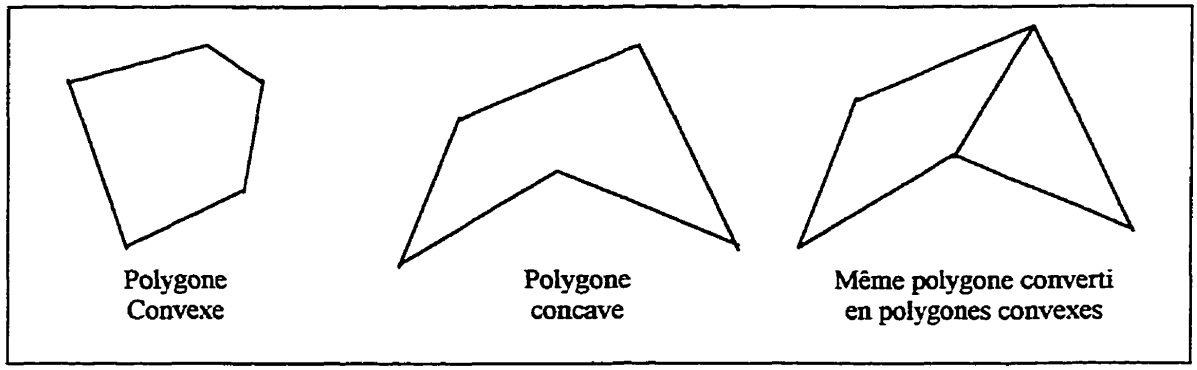


Figure 1.16 Polygone convexe, concave, et polygone converti en polygones convexes

1.5.3 Couleurs des objets

Dans de nombreux logiciels graphiques, les couleurs peuvent être spécifiées directement par trois composantes selon le modèle de couleurs utilisé ou à l'aide d'un index correspondant à un élément d'une palette de couleurs. Mirage possède une palette de 256 couleurs. En appuyant sur la touche « C », on voit apparaître cette palette. Les 16 premières couleurs sont les couleurs de base, soit noir, bleu, vert, cyan, rouge, magenta, brun, gris pale, gris foncé, bleu pale, vert pale, cyan pale, rose, magenta pale et blanc. Le reste de la palette est divisée en 15 dégradés différents.

Dans les fichiers de type PLG, les couleurs sont représentées par un nombre hexadécimal de la forme 0x11FF. Le code 0x signifie que le nombre est en format hexadécimal. Les autres chiffres identifient la couleur.

Il y a 3 types de couleurs dans Mirage, soit les couleurs de base, les couleurs constantes et les couleurs variables:

- Les couleurs 0x0000 à 0x00FF sont les couleurs de base; elles correspondent directement aux couleurs de la palette.
- Les couleurs 0x0100 à 0x0FFF sont les couleurs constantes. Ces couleurs ont une teinte fixe qui ne varie pas avec la direction de la source de lumière (aucun ombrage). Le deuxième chiffre (de 1 à F) identifie la couleur et les deux derniers chiffres représentent l'intensité (00 correspond à l'intensité minimale soit noir, et FF correspond à l'intensité maximale).
- Les couleurs variables vont de 0x1000 à 0x1FFF. La couleur est déterminée avec les trois derniers chiffres de la même façon que les couleurs constantes. La différence est qu'une couleur variable varie en fonction de l'orientation du polygone et de la source de lumière. Comme le montre la Figure 1.17, les seize premières couleurs sont les couleurs de base, les autres couleurs servent à varier l'intensité de couleur des polygones selon l'éclairage.

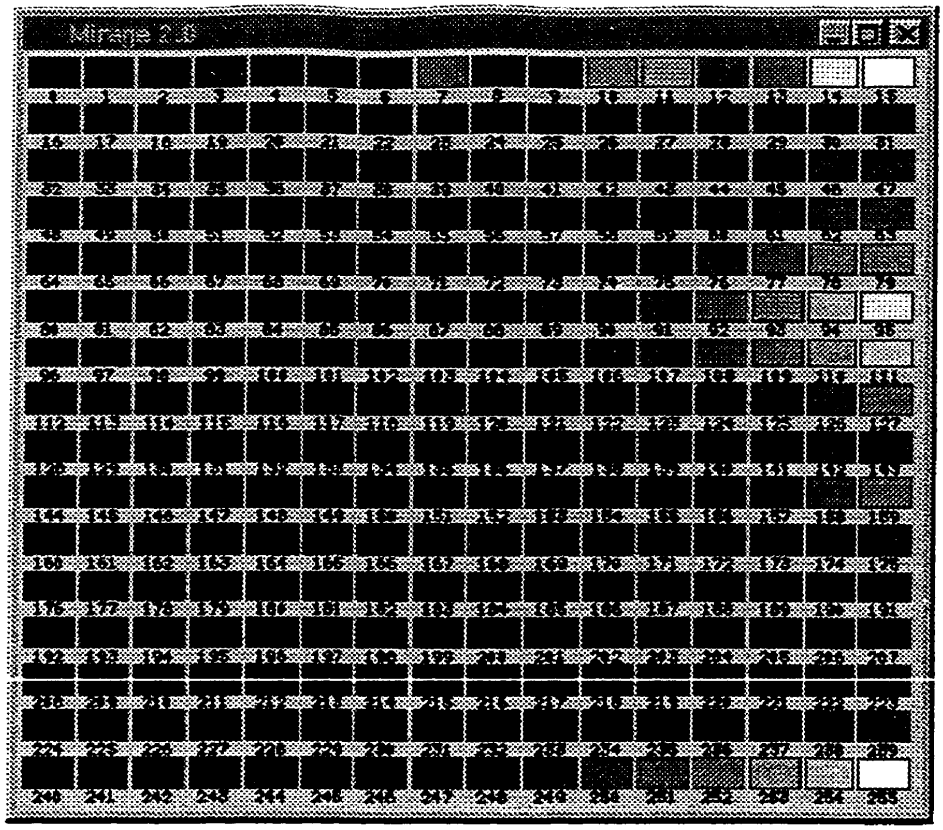


Figure 1.17 Les 256 couleurs de la palette de Mirage

1.5.4 Les fichiers PLG à objets multiples et représentation multiples

Un fichier en format PLG peut contenir plusieurs objets. Ceux-ci sont séparés par leur entête (nom de l'objet, nombre de sommets, nombre de polygones).

Un objet peut avoir plusieurs représentations différentes. Ce principe permet d'avoir différents niveaux de détail (*Level of Detail* ou *LOD*) pour un même objet. Un objet est dessiné ou construit avec un nombre différent de polygones allant d'une représentation simplifiée à complexe (voir Figure 1.18). La représentation sélectionnée lors de

l'affichage à l'écran dépendra de la dimension de l'objet à l'écran en pixels. Une autre possibilité est d'utiliser ce concept pour des animations (objet qui rétrécit, s'agrandit, change de forme, etc...).

Figure 1.18 Niveaux de détail d'une sphère (avec 8 , 48 et 512 polygones)

1.5.5 Le format NFF

Le format NFF provient du logiciel WorldToolKit de la compagnie Sense 8. Les objets en format NFF constituant un monde virtuel peuvent être chargés par Mirage. Leur format est très similaire à celui des fichiers PLG. Les objets peuvent être édités avec un éditeur de texte. Cette fonctionnalité de Mirage est illustrée dans le rapport de stage d'Antoine D'Anjou [D'Anjou 1996]. Un exemple de fichier en format NFF est donné en annexe A (un poste généré par Esope-RV).

1.5.6 Le format DXF

Le format DXF est particulier au logiciel AutoCAD. Il permet tout d'abord d'échanger des fichiers produits à l'aide d'AutoCAD avec d'autres applications et vice-versa. Ces fichiers peuvent être stockés sous forme binaire ou forme ASCII. Mirage permet de charger les fichiers DXF sous forme ASCII [Bélanger 1996].

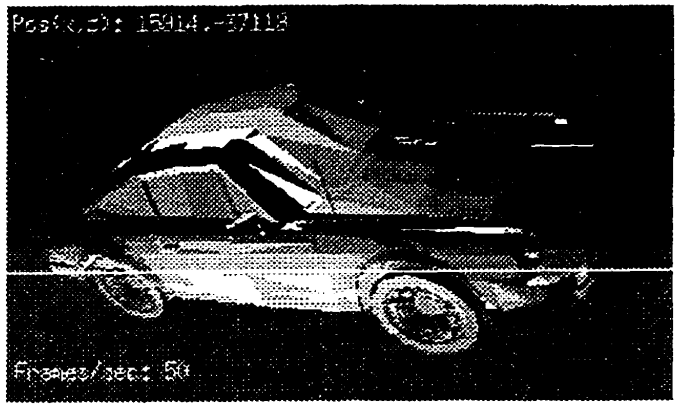


Figure 1.19 Image d'un objet en format DXF

1.5.7 Création des objets

Les objets peuvent être créés de diverses façons. La méthode à utiliser dépend principalement de la complexité de l'objet à créer. Par exemple, un cube peut être construit à l'aide d'un simple éditeur de texte, alors qu'une voiture sera plus facilement créée avec un outil de dessin assisté par ordinateur.

1.5.7.1 Référentiel propre à l'objet et référentiel global

Les coordonnées contenues dans les fichiers de type PLG sont propres à l'objet. Lorsque l'objet est inséré à une position dans le monde virtuel, les coordonnées de l'objet sont modifiées par des translations ou rotations appliquées à l'objet. Les nouvelles coordonnées sont alors dans le référentiel global. Les coordonnées d'un objet peuvent donc être écrites dans deux référentiels: le référentiel de l'objet (coordonnées dans le fichier de type PLG) ou le référentiel global (coordonnées dans le monde virtuel = coordonnées propres de l'objet + coordonnées du point d'insertion de l'objet si aucune rotation est effectuée).

1.5.7.2 Librairies d'objets 3D

Il existe sur l'internet certains sites ayant une librairie d'objets 3D en divers formats. Certains formats peuvent être convertis au format PLG par des logiciels disponibles gratuitement. Parmi ces sites, on trouve UK VR-SIG Object Archive:

[http://www.dcs.ed.ac.uk/~mxr/objets.html].

Le Tableau 1.4 énumère de façon non exhaustive un certain nombre de logiciels de conversion disponibles gratuitement. Ces utilitaires se trouvent sur le site de l'Université de Waterloo [ftp://sunee.uwaterloo.ca/pub/rend386/converters].

Tableau 1.4 Formats différents et convertisseurs au format PLG

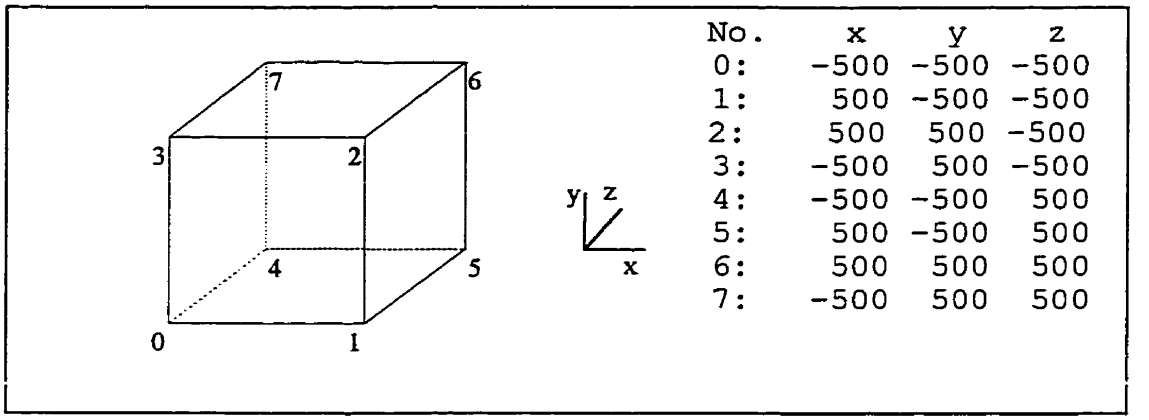
Format du fichier	Application native	Nom du convertisseur
3DS	3D Studio	3ds2plg.exe
BYU	Movie	byu2plg.exe
DXF	AutoCAD	dxf2plg.exe
IRIT	IRIT	irit2plg.exe
OFF		off2plg.exe
SHD	Sun « Shaded »	shd2plg.exe
SRF	SurfModel	srf2plg.exe

1.5.7.3 La création d'objets avec un éditeur de texte

Les objets simples sont généralement construits à l'aide d'un éditeur de texte. Des commentaires peuvent être insérés dans le fichier en utilisant des dièses « # ». Les fichiers de type PLG sont divisés en trois parties :

1. Une entête donnant le nom de l'objet, le nombre de sommets et le nombre de polygones. Ex : `cube 8 6`
2. Une liste des coordonnées de chaque sommet. La numérotation commence à zéro. La première coordonnée sera associée au sommet 0, et ainsi de suite. Ex : `500 -500 500`
3. Une liste de polygones construits en reliant les sommets de la liste précédente. Les polygones sont précédés d'un code de couleur et du nombre de sommets constituant le polygone. Ex : Un polygone de couleur rouge et constitué de 4 sommets : `0x11FF 4 0 1 2 3`

Mirage, comme VR386, utilise un système de coordonnées main gauche. De plus, l'axe des y est dirigé vers le haut.



No.	x	y	z
0:	-500	-500	-500
1:	500	-500	-500
2:	500	500	-500
3:	-500	500	-500
4:	-500	-500	500
5:	500	-500	500
6:	500	500	500
7:	-500	500	500

Figure 1.20 Cube simple et numérotation des points

Un cube pourrait donc avoir la forme suivante dans un fichier `PLG` :

Début du fichier

```
# Un cube visible de
# l'exterieur , 1000x1000
cubeout 8 6

# sommets :
# X    Y    Z
# face avant
-500 -500 -500  # vertex 0
 500 -500 -500  # vertex 1
 500  500 -500  # vertex 2
-500  500 -500  # vertex 3

# face arriere
-500 -500  500  # vertex 4
 500 -500  500  # vertex 5
 500  500  500  # vertex 6
-500  500  500  # vertex 7
```

(suite)

```
# polygones :
# couleur #sommets sommet1 ..

# faces du cube
0x11ff 4 0 1 2 3  # avant
0x12ff 4 7 6 5 4  # arriere
0x13ff 4 2 6 7 3  # haut
0x14ff 4 1 5 6 2  # cote droit

0x15ff 4 0 4 5 1  # bas
0x16ff 4 0 3 7 4  # cote gauche
```

Figure 1.21 Fichier de type `PLG` décrivant un cube

1.5.7.4 La créations d'objets avec un logiciel écrit en C

Une autre façon de générer un objet est de développer un logiciel pour créer le fichier. Par exemple, le logiciel `sphere` génère une sphère avec les paramètre donnés.

Le code source et un exemple de sortie du programme `sphere` sont donnés à l'annexe B. La Figure 1.22 donne un exemple de la sortie de `sphere`.

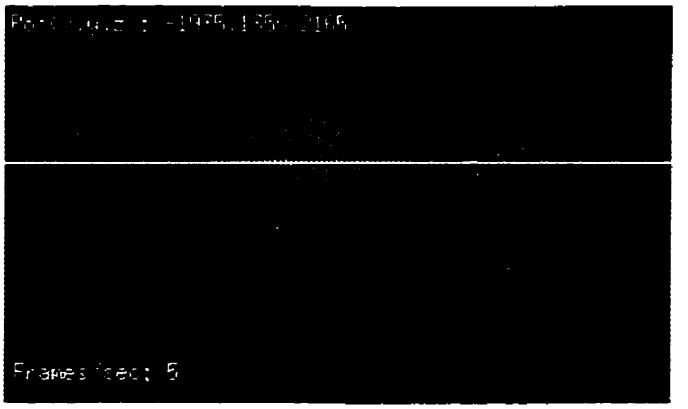


Figure 1.22 Exemple de sortie du programme `sphere`

1.5.7.5 La création d'objets avec des outils de CAO

Les outils de conception assistée par ordinateur (CAO) permettent de créer des objets très complexes. Les logiciels NorthCAD ou AutoCAD peuvent être utilisés pour créer des objets. NorthCAD est présenté dans le livre « Playing God » de Bernie Roehl.

NorthCAD permet d'exporter des fichiers au format PLG, tandis qu'AutoCAD peut générer des fichiers au format DXF, que Mirage peut lire.

1.6 L'utilisation de figures FIG

Les figures sont des regroupements d'objets de type PLG reliés en arborescence. Un objet peut avoir plusieurs objets-enfants mais un seul objet-parent. Une figure peut être vue comme une structure arborescente ou un arbre k-aire (voir Figure 1.23).

Dans l'exemple suivant, une main virtuelle est construite avec des objets de type PLG des fichiers palm.plg, thumb.plg, thtip.plg, finger1.plg et finger2.plg. Les objets enfants sont insérés à certaines positions relatives à l'objet parent et avec des facteurs d'échelle différents pour créer la main. Les doigts sont tous divisés en deux segments ou objets et sont liés à l'objet-parent représentant la paume de la main. La définition de cette main se trouve dans le fichier hand.fig.

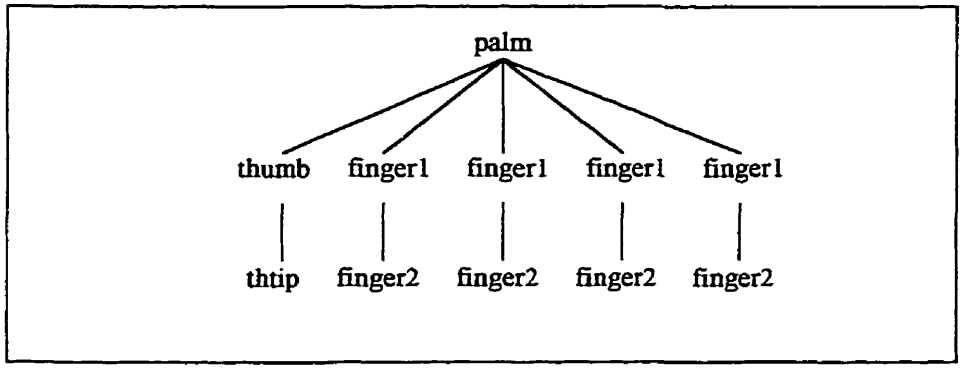


Figure 1.23 Arbre correspondant à la figure représentant une main

L'utilité principale des figures est que chaque transformation affine (translation, rotation, etc..) sur un objet parent se répercute sur les objets enfants. Les figures peuvent donc être

utilisées pour modéliser des objets articulés tel qu'un bras de robot, une figure humaine ou une main comme le montre la figure suivante.

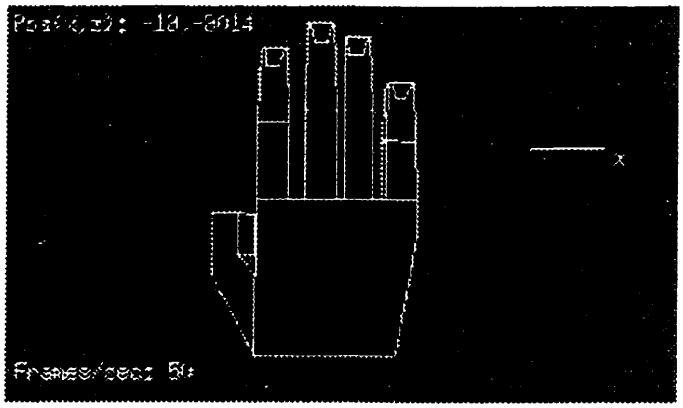


Figure 1.24 Image de la main virtuelle de Mirage (du fichier `hand.fig`)

```
comment = A more complex hand smaller version;
comment = Created by Dave Stampe, July 1992;
comment = Part of the REND386 distribution;

{
  name = palm;
  plgfile = ./palm.plg scale 0.25,0.325,0.25 shift 0,0,0 sort 768;
  pos = 0,-49,0;
  segnum = 0;
  {
    name = thumb;
    plgfile = ./thumb.plg scale 0.25,0.25,0.25 shift 0,0,12 sort 768;
    pos = -64,0,-12;
    rot = 0,30,0;
    segnum = 1;
    {
      plgfile = ./thtip.plg scale 0.25,0.5,0.25 shift -25,0,0 sort 768;
      pos = -40,64,0;
      rot = 0,90,0;
      segnum = 2;
    }
  }
...
```

Figure 1.25 Une partie du fichier `hand.fig` donnant la main virtuelle

1.7 Les fichiers WLD

1.7.1 Création des fichiers WLD

Les fichiers de type WLD décrivent l'ensemble du monde virtuel avec ses objets, ses animations et ses propriétés. Ils sont créés normalement avec des simples éditeurs de texte. Ces fichiers font référence à des fichiers de type PLG, et permettent d'insérer des objets décrits dans les fichiers de type PLG à une position particulière dans le monde virtuel. On peut aussi insérer des figures (fichiers de type FIG). Finalement, on peut spécifier des scénarios d'animation en utilisant les objets ou les figures du monde virtuel ainsi créé. Pour l'instant les fichiers de types NFF et DXF ne peuvent pas être insérés à

partir des fichiers WLD, mais sont intégrés dynamiquement au monde virtuel à l'aide des fonctions du menu décrites précédemment et de la main virtuelle.

1.7.2 Commandes des fichiers WLD

Les commandes du fichier de type WLD ont été créées pour VR386. Ces commandes ont été développées avant Mirage et sont décrites dans plusieurs livres [Stampe 1993]. Les commandes les plus usuelles sont données en annexe. La Figure 1.26 donne un exemple de fichier de type WLD contenant 3 objets.

```
# sphere.wld - Exemple de niveau de detail
# (Level Of Detail example)
# Christophe Maurel, 23 Fevrier 1997

start 0,0,6000 0,180,0 1 # Position de depart, orientation,
                         # zoom

#Sphere avec 512 polygones
object sphere4   1,1,1  0,0,0   -3000,0,0

#Sphere avec 48 polygones
object sphere2   1,1,1  0,0,0    0,0,0

#Sphere avec 8 polygones
object sphere    1,1,1  0,0,0    3000,0,0
```

Figure 1.26 Fichier WLD générant trois sphères

Les noms sphere4, sphere2 et sphere correspondent à des fichiers de type PLG. Les coordonnées qui suivent chaque nom de fichier correspondent respectivement aux facteurs d'échelle (en x, y et z), aux angles de rotation (relatifs aux axes des x, y et z) et à la position (x, y et z) à donner à l'objet. Les paramètres de départ de la caméra sont donnés après la commande start, soit respectivement sa position, son orientation (relative

aux axes des x, y et z) et son facteur d'agrandissement [*zoom*]. Une liste des commandes principales des fichiers `WLD` est donnée en annexe.

1.7.3 Conversion des fichiers `WLD` à VRML

Les fichiers `WLD` peuvent être convertis au format VRML (*Virtual Reality Modelling Language*), un langage mis au point pour décrire des mondes virtuels et à l'aide d'un visualisateur approprié, visualiser ces mondes virtuels sur l'internet [Ames 1996]. Un utilitaire de conversion développé par Bernie Roehl est disponible gratuitement sur l'internet à l'Université de Waterloo:

`[ftp://sunee.uwaterloo.ca/pub/rend386/converters]`

Cet utilitaire est appelé `wld2vrml` et fonctionne sous DOS. Ces fichiers d'extension `WRL` peuvent ensuite être visualisés par un logiciel tel que Live3D de Netscape ou Cosmo de Silicon Graphics.

1.8 Ajouter des textures

Il peut être utile d'appliquer des textures sur certains polygones. Les textures [Blinn 1976] doivent être utilisées judicieusement car elles ralentissent l'affichage des mondes virtuels [Crow 1984]. Dans Mirage, les textures sont contenues dans des fichiers de type `PCX`. Ces images en pixels [*bitmaps*] peuvent être créées avec un simple logiciel de dessin

comme Paintbrush[1]. Certains programmes de dessin ou sites internet contiennent des textures préfabriquées.

Figure 1.27 Texture (logo SGI) appliquée sur un polygone de Mirage

Le fichier `textures.map` contient la liste des textures disponibles. Ce fichier est extensible.

```
0  monalis.pcx
1  sgilogo.pcx
2  panther.pcx
```

Figure 1.28 Une partie du fichier textures.map

Le numéro se réfère à l'index correspondant à la texture. Au lieu d'avoir un code de couleur, les polygones texturés ont un index de texture.

[1] Paintbrush est un outil de dessin fourni avec Windows.

1.9 Résumé

Dans ce chapitre, nous avons donné l'ensemble des éléments pour décrire un monde virtuel et décrit l'interface de Mirage. Dans le chapitre suivant, nous expliquons le développement d'applications avec Mirage.

Chapitre 2. Développer une application avec Mirage

Dans le chapitre précédent, nous avons décrit l'interface de Mirage et ses caractéristiques générales ainsi que la création de mondes virtuels animés au moyen des fichiers de type WLD. Dans le présent chapitre, nous expliquons le développement de logiciels en langage C à l'aide d'appels à des fonctions des bibliothèques graphiques de Mirage. Cette façon de procéder est assez laborieuse mais a l'avantage de donner une plus grande flexibilité. Cependant, avant de parler des fonctions de Mirage utiles au développement d'applications en réalité virtuelle, nous présentons tout d'abord les principales structures de données de Mirage.

2.1 Représentation interne d'un monde virtuel

La représentation interne d'un monde virtuel dans Mirage (voir Figure 2.1) peut être divisée en deux parties:

- un ensemble de structures de données qui décrivent la partie statique du monde virtuel (y compris les objets);
- une bibliothèque de fonctions qui permettent de modifier ces structures de données, créant un monde virtuel dynamique (avec du mouvement et une certaine interactivité).

Comme les structures de données et la bibliothèque de fonctions sont écrites en langage C, le développement d'applications avec Mirage requiert une bonne connaissance de ce langage, qui est traité dans de nombreux ouvrages [Kelley 1990]. En particulier, une

bonne notion des pointeurs est importante pour utiliser les fonctions d'applications de Mirage. Également, une connaissance moyenne de X Windows facilite le développement. Des notions de base sur X Windows sont données dans le prochain chapitre.

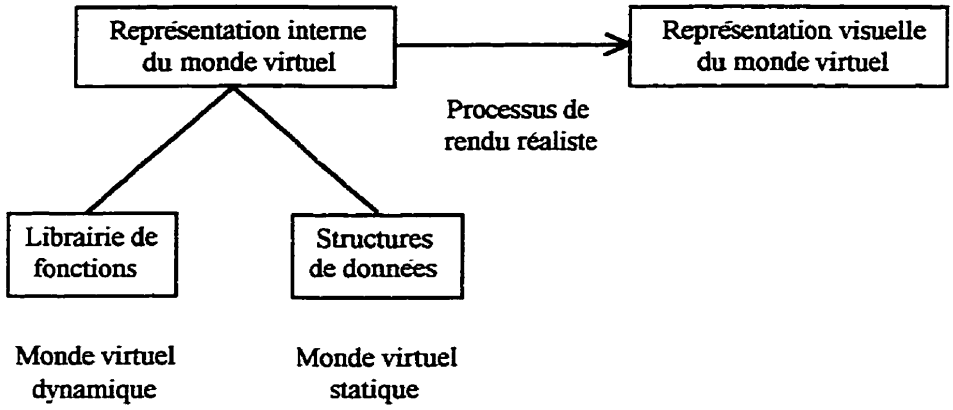


Figure 2.1 Représentations interne et visuelle d'un monde virtuel de Mirage

Comme le montre la Figure 2.1, le passage de la représentation interne à la représentation visuelle, soit des structures de données à l'affichage à l'écran du monde virtuel, est effectuée dans le processus de rendu réaliste [*rendering*], qui est expliqué dans le chapitre suivant.

2.2 Les structures de données de Mirage

Les données du monde virtuel sont organisées dans des structures de données en langage C. Cette organisation est nécessaire dans tout logiciel d'envergure tel que Mirage. Sa

principale utilité est de permettre l'acquisition d'une vue globale des données et donc de faciliter l'accès et la modification de ces données. Chaque structure de données a un type particulier qui englobe des éléments d'information (également appelés champs) reliés au même concept. Afin de bien comprendre le développement d'applications avec Mirage, il est nécessaire d'expliquer les types des structures de données utilisées par Mirage.

2.2.1 Les structures de données associées aux objets

Ces structures décrivent un objet dans Mirage, ou servent à modifier des objets. Les relations entre ces types sont illustrées dans la figure suivante:

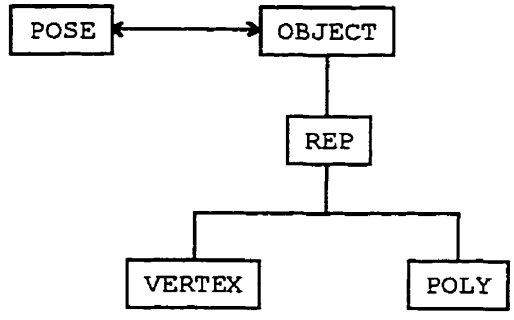


Figure 2.2 Les types des structures associées aux objets

Les objets de Mirage sont décrits dans une structure de type `OBJECT`. Cette structure fait référence à une autre structure de type `REP` (représentation d'un objet), qui fait référence elle-même aux structures de type `POLY` (liste des polygones de l'objet) et

VERTEX (liste des sommets des polygones). La structure POSE sert principalement d'intermédiaire dans la modification des positions et orientations des objets.

2.2.1.1 Le type POSE

Il est important de décrire le type POSE, qui sert dans de nombreuses fonctions. Le type POSE est défini de la façon suivante:

```
typedef struct  {
   COORD x,y,z;          // notation 32.0
   ANGLE rx,ry,rz;       // notation 16.16
} POSE;
```

Les types COORD et ANGLE sont eux-mêmes définis comme étant de type `long`, soit des entiers de 32 bits. Le type POSE contient donc trois coordonnées `x, y` et `z` qui sont des entiers. Mirage utilise des coordonnées entières. Trois angles de rotation `rx, ry` et `rz` sont aussi contenus dans le type POSE. Ces angles sont en notation point fixe 16.16 c'est-à-dire que les 16 premiers bits de l'entier contiennent la partie entière de l'angle, et les 16 derniers bits contiennent la partie fractionnaire. Afin de convertir un angle en degrés en notation 16.16 interne à Mirage, il suffit de le multiplier par 2^{16} soit 65536. Il est possible d'avoir une perte de précision lors de cette conversion. A l'inverse, pour obtenir l'angle en degrés à partir de la valeur 16.16, il faut diviser par 65536.

Le type POSE peut servir à décrire la position et l'orientation d'un objet ou d'une caméra dans Mirage.

2.2.1.2 Le type OBJECT

La structure de type OBJECT est utilisée pour conserver les données des objets de Mirage. La définition du type est la suivante:

```
typedef struct _object
{
 unsigned int oflags;     /* attributs de l'objet */
 struct _object *prev;    /* pointeur à l'élement préc */
 struct _object *nnext;   /* pointeur à l'élément suivant */
 void *owner;             /* par ex., struct segment body */
 REP *replist;          /* point. liste de représentations */
 REP *current_rep;        /* représentation active  */
 long osphx, osphy, osphz;  /* centre sphere ref. objet */
 long sphx, sphy, sphz, sphr; /* centre, rayon sphère */
 long minx, miny, minz, maxx, maxy, maxz;/* bounding cube*/
 long update_count;       /* inc. si objet est déplacé */
} OBJECT;
```

Le premier champ oflags contient des informations sur le type et l'état de l'objet. Les bits de ce champ indiquent certaines caractéristiques de l'objet (ex: objet sélectionnable ou non-sélectionnable). Deux pointeurs de type OBJECT * servent à créer des listes doublement chaînées d'objets (voir Figure 2.3). Les listes doublement chaînées permettent d'effectuer des insertions et retraits rapides d'éléments. Elles sont donc utilisées pour améliorer la performance. Le pointeur prev indique l'objet précédent dans la liste, tandis que l'élément suivant est donné par le champ nnext comme le montre la Figure 2.3.

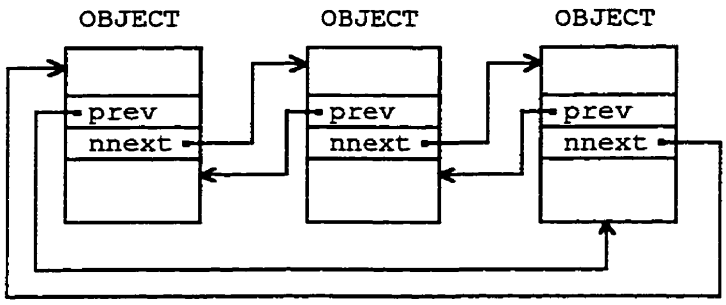


Figure 2.3 Une liste doublement chaînée d'objets

Le champ `owner` sert à identifier l'objet parent s'il existe et si l'objet fait partie d'une figure.

Deux pointeurs de type `REP *` sont présents dans la structure `OBJECT`. Le premier pointeur `replist` indique le premier élément dans la liste des représentations de l'objet. Chaque objet possède donc une liste de représentations (voir Figure 2.4).

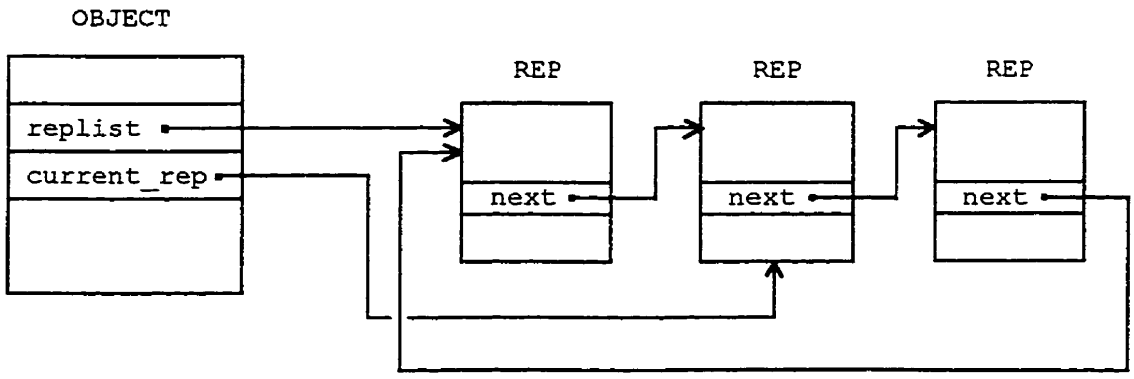


Figure 2.4 Un objet avec trois représentations

Souvent cette liste ne contient qu'un seul élément (représentation unique, voir Figure 2.5). Le pointeur `current_rep` donne la représentation active, soit celle qui sera présentée à l'écran.

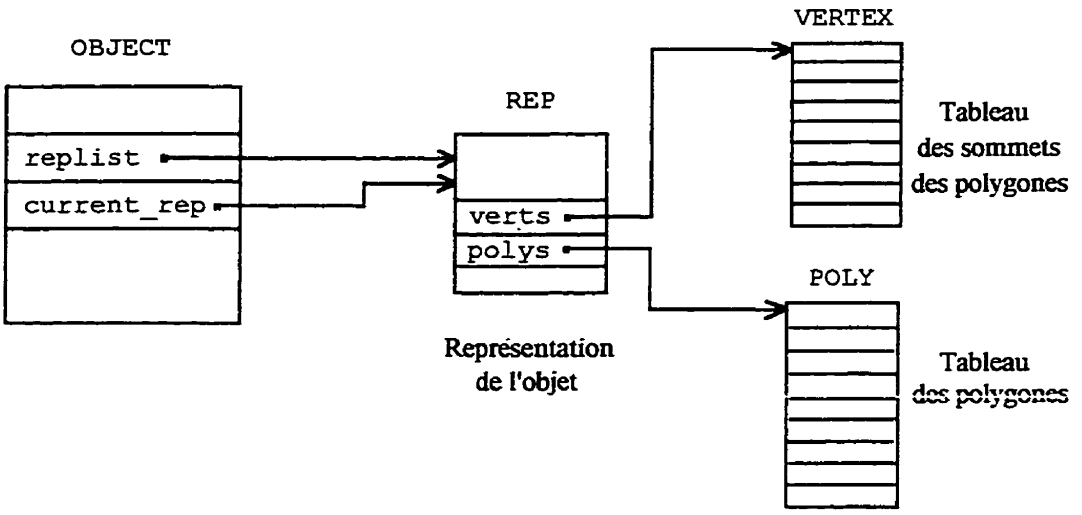


Figure 2.5 Un objet typique avec une seule représentation

Chaque objet possède une sphère englobante qui est la sphère de rayon minimal englobant l'objet au complet. Cette sphère sert à détecter les collisions entre les objets (ex: une main virtuelle). Les champs `osphx`, `osphy` et `osphz` sont les coordonnées du centre de la sphère dans le référentiel de l'objet. Les champs `sphx`, `sphy`, `sphz` et `sphr` sont les coordonnées et rayon de la sphère dans le référentiel global.

Le dernier champ `update_count` sert dans les animations. Cette variable est incrémentée et testée pendant les animations.

2.2.1.3 Le type REP

Le type REP est utilisé pour décrire les objets de Mirage. Chaque représentation possède un champ size, qui est la dimension minimale en pixels de l'objet pour que cette représentation soit choisie.

```
typedef struct _rep {   /* une representation d'objet */
  int size;  /* si l'object est + grand, utiliser rep */
  int nverts, npolys;     /* nb de sommets, nb de polys */
  VERTEX *verts;          /* tableau des sommets */
  POLY *polys;            /* tableau des polygones */
  struct _rep *next;      /* pointeur a prochaine rep */
  long update_count;      /* inc. avec chaque déplacement */
  unsigned flags;
} REP;
```

Les champs nverts et npolys correspondent respectivement aux nombres de sommets et de polygones dans la représentation. Le champ verts pointe au tableau des sommets de l'objet. Les sommets sont utilisés pour créer les polygones. Le tableau contient la liste des coordonnées de ces sommets. Le champ polys pointe au tableau des polygones de l'objet. Ce tableau contient des éléments de type POLY. Chaque polygone est spécifié par une liste d'index correspondant à des sommets dans le tableau des sommets.

L'élément suivant dans la liste de représentations est donné par le champ next. Le champ update_count est incrémenté et testé lors des animations et le champ flags indique l'état de l'objet.

2.2.1.4 Le type VERTEX

Ce type sert à stocker les sommets d'un polygone. Les coordonnées d'un sommet sont spécifiées dans deux référentiels: le référentiel de l'objet (`ox,oy,oz`) et le référentiel global (`x, y, z`). Les autres champs sont internes et ne devraient pas être modifiés car ils sont recalculés à chaque mise à jour de l'écran. Par exemple, le champ `cz` contient la coordonnée z convertie ou profondeur du sommet par rapport au point de vue, et est utilisé pour l'élimination des faces arrières.

```
typedef struct {
 long ox, oy, oz;    /* coordonnées de l'objet */
 long x, y, z;       /* coordonnées  globales */
 long cz;           /* coord Z convertie */
 NVERTEX *new_copy;  /* non-null si transformés */
 unsigned char z_transformed; /* non-null z transformé */
 unsigned char z_outcode;    /* 1 si hither, 2 si yon */
} VERTEX;
```

2.2.1.5 Le type POLY

Le type `POLY` contient la description d'un polygone. La définition est la suivante:

```
typedef struct {
 unsigned color;    /* couleur (non-utilisée) */
 VERTEX **points;   /* tableau de pointeurs aux vertex */
 int npoints;       /* nombre  d'entrées dans points[] */
 long onormalx, onormaly, onormalz; /* vecteur normal */
 long normalx, normaly, normalz;  /* vecteur normal glob.*/
 struct _object *object;
} POLY;
```

A chaque polygone est associé une couleur (champ `color`). Le pointeur `**points` donne l'adresse d'un tableau de pointeurs aux sommets du polygone. Le nombre de points du polygone est donné par le champ `npoints`. Le vecteur normal unitaire du polygone est donné dans deux référentiels: le référentiel de l'objet (`onormalx, onormaly,onormalz`) et le référentiel global (`normalx,normaly,normalz`). Le vecteur normal est utilisé pour les calculs de visibilité et de teinte du polygone. Finalement, le champ `object` donne l'adresse de l'objet auquel appartient le polygone.

2.2.2 Les structures de données associées à la visualisation du monde virtuel

La vue de l'usager du monde virtuel dépend principalement d'une structure de type `VIEW`. Cette vue est modifiée par l'intermédiaire des structures `TELEPORT` et `CAMERA`. Le type `POSE`, décrit dans la section précédente, sert également dans la modification des structures de types `TELEPORT` et `CAMERA`.

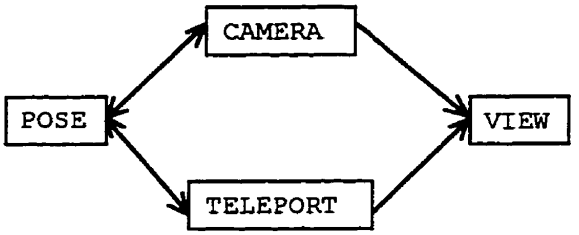


Figure 2.6 Les types des structures associées à la visualisation du monde virtuel

2.2.2.1 Le type VIEW

Le type VIEW contient de nombreux champs associés au point de vue. La définition est la suivante:

```
typedef struct {
 long zoom;      /* <16.16> 1/tan(H FOV/2) 0.5 a 16  */
 /* DONNEES VUE */
 long left,right;         /* <25.0> plans découpage   */
 long top, bottom;
 long hither, yon; /* <25.0> découpage proche, loin  */
 long aspect;       /* <16.16> facteur x:y (mag. Y) */
 /* CONTROLE DU RENDU */
 unsigned flags;               /* 16 bits de flags   */
 /* DONNEES DE RENDU ADVANCEES */
 long x_offset, y_offset;  /* déplacement du centre */
 unsigned orientation;     /* pour image miroir     */
 MATRIX eye_xform;
  /* DONNEES INTERNES */
 long hsw, hsh;          /* moitiés de width, height     */
 long hsc, vsc;          /* centre de l'image (offset)   */
 long scx, scy;
 long sx,sy;
 int  xshift, yshift;
 long left_C, left_M;    /* coefficients clipping sphèr.*/
 long right_C, right_M;
 long top_C, top_M;
 long bot_C, bot_M;
} VIEW;
```

La grande majorité des champs de ce type sont internes et n'ont pas à être modifiés. Ces champs contiennent des informations relatives à la dimension de la fenêtre X Windows, aux plans de découpage [*clipping*], et au point de vue de la caméra donnant la vue sur le monde virtuel. Il existe de nombreuses fonctions de la bibliothèque qui modifient ces champs.

2.2.2.2 Le type CAMERA

Le type CAMERA est défini de la manière suivante:

```
typedef struct {
  VIEW *mono;        // vue centrale/mono
  VIEW *left;        // vue oeil gauche
  VIEW *right;       // vue oeil droit
  BOOL is_stereo;    // vue stéreo ?
  STEREO *stereo;    // sert pour créer vues stereo
  SEGMENT *seg;      // segment pour déplacer caméra
} CAMERA;
```

Le type CAMERA contient un pointeur `mono` de type `VIEW *` servant pour la vue monoscopique. Les champs suivants `*left`, `*right`, `is_stereo` et `*stereo` servent à la vue stéréoscopique, qui n'est pas implantée dans Mirage. Le dernier champ `*seg` sert à rattacher la caméra à un objet.

Notons que la variable globale `current_camera` est presque toujours donnée en argument aux fonctions associées aux caméras.

2.2.2.3 Le type TELEPORT

Le type TELEPORT est également utilisé dans Mirage. Ce type est défini de la façon suivante:

```
typedef struct {
  POSE current_pose;
  OBJECT *vehicle;
  char * vehicle_name;
  WORLD *world;
} TELEPORT;
```

Le type TELEPORT sert généralement pour se déplacer d'un point à un autre. Un point de téléportation est similaire à une position de caméra mais contient des champs supplémentaires. Le champ `current_pose` contient la position de la caméra et son orientation spatiale. Le champ `*vehicle` permet d'attacher un point de téléportation à un objet qui se déplace. Finalement, le champ `*world` permettrait de se téléporter entre des mondes virtuels différents mais cette capacité n'existe pas dans Mirage. Ce champ n'est donc pas utilisé, comme les champs associés à la vue stéréoscopique du type CAMERA.

2.3 La bibliothèque de fonctions de Mirage

Un principe de « bonne » programmation est que les champs des structures de données ne devraient pas être modifiés directement. Il est préférable d'utiliser des fonctions existantes ou de créer ses propres fonctions qui modifient les champs. Cette pratique fait partie des méthodes de la programmation orientée-objet.

Les fonctions énumérées ne font pas partie d'une liste exhaustive. Nous présentons uniquement les fonctions les plus utiles. Les fonctions sont données avec leurs prototypes accompagnés d'une brève description.

2.3.1 Les fonctions associées aux objets

Ces fonctions ont souvent au moins un argument de type `OBJECT *` soit un pointeur à une variable de type `OBJECT`. Ce pointeur permet de modifier la structure interne des objets. Ces fonctions servent donc à créer, déplacer les objets, à modifier ou récupérer leurs attributs. Les paramètres et les valeurs de retour des fonctions sont détaillées à l'annexe C.

```
void add_object_to_world(OBJECT *obj)
```

Comme son nom l'indique, cette fonction rajoute un objet au monde virtuel. Même si un objet est chargé avec la fonction `load_plg_object`, il n'apparaît dans le monde que si les fonctions `update_object` et `add_object_to_world` sont appelées.

```
void attach_object(OBJECT *obj, OBJECT *to, BOOL preserve)
```

Cette fonction attache l'objet donné par `*obj` à un autre objet donné par `*to`. Cet objet `*to` devient parent de l'autre.

```
void compute_object(OBJECT *)
```

Les paramètres de l'objet sont mis à jour.

```
void detach_object(OBJECT *obj, BOOL preserve)
```

L'objet donné est détaché de son parent, s'il existe. Si l'objet n'a aucun parent, la fonction n'a aucun effet

```
void do_for_all_objects(void (*fn)(OBJECT *))
```

La fonction `fn` est appelée pour tous les objets contenus dans le monde virtuel.

```
void do_for_all_selected(void (*fn)(OBJECT *))
```

La fonction `fn` est appelée pour tous les objets sélectionnés.

```
OBJECT *find_object_on_screen(WORD x, WORD y)
```

Cette fonction est utile pour trouver quel objet a été sélectionné avec la souris. Elle recherche un objet aux coordonnées `x, y` de la fenêtre de Mirage.

```
long get_object_bounds(OBJECT *obj, long *x, long *y, long
*z)
```

Afin de faire des tests de collision entre objets, on peut utiliser la sphère englobante de l'objet. Cette fonction retourne le rayon et les coordonnées du centre de la sphère via les pointeurs `x, y` et `z`.

```
void get_obj_info(OBJECT *obj, int *nv, int *np)
```

Cette fonction récupère le nombre de sommets et de polygones contenus dans l'objet donné en argument.

```
void get_object_pose(OBJECT* obj, POSE *p)
```

Cette fonction est utile pour obtenir la position et les angles de rotation d'un objet dans le référentiel de l'objet.

```
void get_object_world_pose(OBJECT *obj, POSE *p)
```

Cette fonction place dans une variable de type `POSE` la position et les angles de rotation d'un objet dans le référentiel global.

```
void get_object_world_position(OBJECT *obj, long *x, long
*y, long *z)
```

Cette fonction place dans les variables `x, y` et `z` la position de l'objet spécifiée dans le référentiel global.

```
void highlight_object(OBJECT *obj)
```

L'objet donné en argument est mis en évidence en traçant en blanc les bordures de ses polygones pour montrer qu'il est sélectionné.

```
BOOL is_object_selectable(OBJECT *obj)
```

La fonction retourne une valeur 1 si l'objet peut être sélectionné. Un objet représentant un curseur 3D est un exemple d'objet qui ne devrait pas être sélectionné.

```
BOOL is_object_selected(OBJECT *obj)
```

La fonction retourne 1 si l'objet a été sélectionné, sinon 0.

```
OBJECT *load_plg_object(FILE *in, POSE *p, float sx, float
sy, float sz, WORD depth)
```

Cette fonction charge un objet de type PLG à partir d'un fichier. Le fichier doit être préalablement ouvert avec la fonction `fopen()` en mode lecture. La position et l'orientation sont données par le paramètre `p`. Les valeurs `sx`,`sy` et `sz` sont les facteurs d'échelle.

```
OBJECT *load_nff_object(FILE *in, POSE *p, float sx, float
sy, float sz, WORD depth)
```

La fonction charge un objet de type NFF. Voir la fonction précédente pour plus d'informations.

```
void remove_object_from_world(OBJECT *obj)
```

Cette fonction retire un objet du monde virtuel. L'objet n'est pas effacé de la mémoire. Il peut donc être rappelé ultérieurement avec `add_object_to_world`.

```
void save_plg(OBJECT *obj, FILE *out, BOOL world)
```

Cette fonction sauvegarde un objet dans un fichier PLG. Le fichier doit être préalablement ouvert avec la fonction `fopen()` en mode écriture. Les coordonnées seront soit dans le référentiel de l'objet, soit dans le référentiel global selon l'argument `world`.

```
void select_next_representation(OBJECT *obj)
```

La représentation suivante de l'objet est activée. Si la représentation active est la dernière, la première représentation est choisie.

```
void set_object_pose(OBJECT *obj, POSE *p)
```

Cette fonction spécifie la position et l'orientation d'un objet dans le référentiel de l'objet. Pour que les changements soient visibles, on doit appeler la fonction `update_object`.

`void set_object_world_pose(OBJECT *obj, POSE *p)`

Identique à la fonction précédente, sauf que le référentiel global est utilisé.

`void unhighlight_object(OBJECT *obj)`

Cette fonction désélectionne un objet. Les bordures des polygones de l'objet ne seront plus tracées en blanc.

`void update_object(OBJECT *obj)`

Cette fonction fait la mise à jour de l'objet. Elle est généralement appelée après une fonction du type `set_object_pose`.

2.3.2 Exemples d'utilisation de la bibliothèque

2.3.2.1 Chargement d'objets contenus dans un fichier PLG

Dans le module `init.C`, Mirage vérifie si un fichier de type `PLG` a été spécifié en argument au lancement. Si c'est le cas, le fichier est ouvert avec la fonction `fopen()`. L'objet est ensuite chargé à la position 0,0,0 avec des angles de rotation de 0,0,0 (`ZERO_POSE`). Les facteurs d'échelle en `x, y, z` à appliquer à l'objet sont de 1,1,1. Un fichier de type `PLG` peut contenir plusieurs objets. La fonction `load_plg_object` est donc appelée tant que tous les objets n'ont pas été lus (voir Figure 2.7).

```
OBJECT *obj;
POSE p = ZERO_POSE;
while ((obj = load_plg_object(in, &p, 1, 1, 1, 0)) != NULL)
{
  if (make_fixed_object_moveable(obj,NULL) == NULL)
     errprintf("Warning: out of memory while loading an object\n");
   else
   {
      set_object_pose(obj,&p);
      update_object(obj);
      add_object_to_world(obj);
   }
}
```

Figure 2.7 Chargement d'un objet PLG

2.3.2.2 Sélection d'un objet à l'écran

Dans cet exemple tiré du fichier `Xmouse.C`, l'usager a désigné avec la souris l'intérieur de la fenêtre de Mirage. On vérifie si un objet se trouve à l'endroit désigné et on met en évidence l'objet avec la fonction `highlight_object` ou s'il est déjà sélectionné on enlève cette mise en évidence. La fonction reçoit une variable de type `Xevent (report)` qui contient les coordonnées du curseur dans les champs `report.xbutton.x` et `report.xbutton.y`. Voir la Figure 2.8.

```
OBJECT *obj;

obj = find_object_on_screen( report.xbutton.x,  report.xbutton.y);

 if (obj && is_object_selectable(obj)) // a-t-on trouvé un objet
sélectionnable à cet endroit ?
 {
    if(is_object_selected(obj))  // l'objet est-il déjà
                                 // sélectionné ?
     unhighlight_object(obj);
    else
      highlight_object(obj); // mettre en évidence l'objet
    world_changed++;  // s'assurer que l'image sera rafraichie
  }
  else
    popmsg("Not on (selectable) object"); // Afficher qu'aucun
                                          //objet n'a été trouvé
```

Figure 2.8 Sélection d'un objet à l'écran

2.3.2.3 Développement d'une application

Les scénarios d'animations contenus dans les fichiers de type `WLD` sont traités en deux étapes:

- initialisation : chargement des objets et lecture des scénarios d'animation (fonction `read_cfg()`);
- mise à jour continuelle du monde virtuel en prenant en compte les scénarios d'animation (fonctions `run_tasks()` et `do_animations()`).

Le même concept peut être repris pour développer des scénarios d'animation directement en langage C:

- initialisation: chargement des objets (fonction `mirage_init()`);

- mises à jour du monde virtuel selon les scénarios d'animation (fonction `mirage_update()`).

Deux fonctions, `mirage_init()` et `mirage_update()`, font partie du fichier `mirage.C` (voir Figure 2.10) et sont initialement vides. La fonction `mirage_init()` est appelée une seule fois au démarrage de l'application. La fonction `mirage_update()` est appelée à chaque itération dans la boucle principale de Mirage. Cette boucle est illustrée au chapitre suivant.

Voici donc un exemple d'animation. Dans `mirage_init()`, un cube est d'abord chargé à partir d'un fichier de type `PLG` et placé dans le monde virtuel. Dans la fonction `mirage_update()`, on applique plusieurs rotations au cube afin de donner au cube un mouvement pseudo-aléatoire. On obtient alors une animation d'un cube tournant illustré dans la Figure 2.9. La même animation peut être effectuée par un fichier de type `WLD` (voir annexe E).

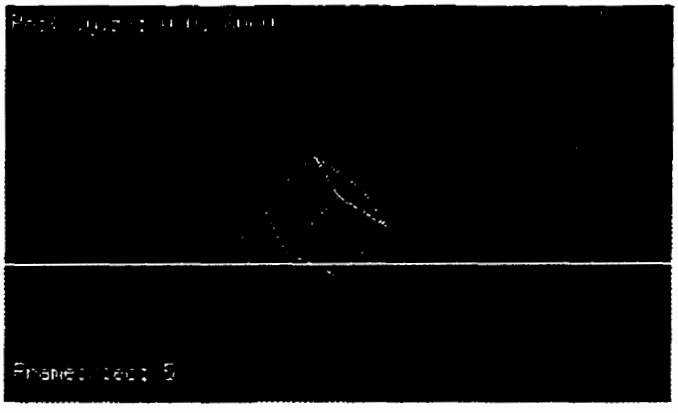


Figure 2.9 Vue de l'animation d'un cube tournant

Après chaque modification dans le fichier `mirage.C` (ou tout autre fichier C), on doit recompiler Mirage avec la commande `make`. On obtient alors un nouvel exécutable `mirage`.

```
// mirage.C
// Christophe Maurel
// 16 Décembre 1996

#include <stdio.h>
#include "vr_api.h" // contient les prototypes des fonctions de Mirage

OBJECT *cube;  // pointeur a l'objet cube
// Position et orientation du cube
// x,y,z = 0,50,0; rx,ry,rz = 0,0,0
POSE pose_cube = { 0,50,0, 0,0,0 };

void mirage_init()
{
  char nomfichier[20] = "plg/block.plg";
  FILE *fichier;

  // charger le cube
  if ((fichier = fopen(nomfichier,"r")) == NULL)
  {
    printf("Erreur de lecture du fichier %s\n\n",nomfichier);
    exit(1);
  }
  cube = load_plg_object(fichier,&pose_cube,10,10,10,0);
  if (cube)
  {
    if (make_fixed_object_moveable(cube,NULL) != NULL)
    {
      set_object_pose(cube,&pose_cube); // positionne le cube
      update_object(cube);              // mise à jour de l'objet
      add_object_to_world(cube);        // rajoute objet au monde
    }
  }
  else
  {
    printf("Erreur en chargeant le cube\n\n");
    exit(1);
  }
}
// Appliquer des rotations du cube par rapport aux axes x,y et z
void mirage_update()
{
  extern int world_changed;

  get_object_pose(cube,&pose_cube);  // lire la pose du cube
  pose_cube.rx += 4 * 65536L;
  pose_cube.ry += 10 * 65536L; // rotation par boucle: 4,10 5 d.
  pose_cube.rz += 5 * 65536L;  // mouvement pseudo-aléatoire
  set_object_pose(cube,&pose_cube);  // modifier la pose du cube
  update_object(cube);        // mise à jour de l'objet
  world_changed++;           // on fera la mise à jour de la fenêtre
}
```

Figure 2.10 Listage du fichier mirage.C contenant une animation de cube tournant

Malheureusement il est difficile de combiner le développement d'animations en C et des mondes virtuels décrits en format WLD car les structures de données des fichiers de type WLD sont difficilement accessibles via le langage C. Il est donc recommandé d'utiliser une approche ou l'autre. Les avantages et inconvénients de chaque méthode sont donnés dans les tableaux suivants:

Tableau 2.1 Avantages et inconvénients des scripts WLD

Avantages	Inconvénients
• programmation aisée; • possibilité de convertir au format VRML (mais sans les animations).	• limité par les fonctions WLD; • difficulté de porter les animations à un autre logiciel RV; • interactivité limitée avec l'usager; • contrôle limité de la simulation.

Tableau 2.2 Avantages et inconvénients du développement en langage C

Avantages	Inconvénients
• en introduisant de nouvelles fonctions, les possibilités sont presque illimitées; • possibilité de porter les animations à un autre logiciel RV; • grande interactivité possible avec l'utilisateur; • la simulation peut être bien contrôlée par le programme ou même à distance par l'utilisation de sockets de TCP/IP [Ahuja, Clouâtre 1996].	• programmation plus complexe; • difficile de convertir au format VRML.

2.3.3 Les fonctions associées aux caméras et téléportations

Ces fonctions permettent de modifier la vue du monde virtuel, soit par les caméras ou les téléportations. Les modifications pouvant être effectuées sont entre autres des changements de la position de la caméra et des effets de loupe.

```
compute_camera_factors(CAMERA *c)
```

Cette fonction fait la mise à jour des paramètres associés aux caméras. Elle doit être appelée après `set_camera_window` par exemple.

```
TELEPORT *create_teleport()
```

La fonction alloue un espace mémoire pour une variable de type `TELEPORT` et l'initialise.

```
void delete_teleport(TELEPORT *)
```

Cette fonction libère la mémoire occupée par une variable de type TELEPORT. Elle est utile uniquement si la variable a été allouée de façon dynamique.

```
void get_camera_worldpose(CAMERA *c, POSE *p)
```

Cette fonction récupère la position d'une caméra dans le référentiel global ainsi que ses angles de rotation.

```
SCALE get_camera_zoom(CAMERA *c)
```

Cette fonction récupère la valeur de l'effet de loupe de la caméra donnée.

```
void set_camera_hither(CAMERA *c, COORD h)
```

Cette fonction spécifie la distance par rapport au plan avant du parallélépipède de vision. Les objets à l'avant du plan avant ne seront pas affichés. Le découpage [*clipping*], qui fait partie du processus de rendu réaliste [*rendering*], est expliqué au chapitre 3.

```
void set_camera_window(CAMERA *c, WORD l, WORD t, WORD r,
WORD b)
```

Cette fonction crée une fenêtre de visualisation à l'intérieur de la clôture de Mirage (*viewport*).

```
void set_camera_yon(CAMERA *c, COORD y)
```

Cette fonction ajuste la distance du plan arrière du parallélépipède de vision. Les objets à l'arrière du plan arrière ne sont pas affichés. Le découpage [*clipping*], qui fait partie du processus de rendu réaliste [*rendering*], est expliqué au chapitre 3.

```
void set_camera_zoom(CAMERA *c, SCALE zoom)
```

Cette fonction modifie l'effet de loupe de la caméra donnée. Par défaut cette valeur est de 1.

```
void teleport_set_here(TELEPORT *t, POSE *p)
```

Cette fonction sauvegarde un point de téléportation. Pour sauvegarder la position actuelle, on donne la variable `current_pose` en argument.

```
void teleport_set_vehicle(TELEPORT *t, OBJECT *vehicle,
char *vname)
```

Le point de téléportation est attaché à un objet véhicule. Si l'objet se déplace, le point de téléportation suivra.

```
void teleport_to(TELEPORT *t)
```

Cette fonction téléporte l'usager au point de téléportation donné.

2.4 La compilation, l'optimisation et le déboguage

Tout programme UNIX comportant de nombreux fichiers C ou C++ nécessite forcément un fichier `makefile`, soit un fichier donnant la liste des fichiers contenant le code source, ainsi que les instructions et options de compilation. Le `makefile` de Mirage, ainsi que l'optimisation et le déboguage d'un programme UNIX tel que Mirage sont montrés en annexe F.

2.5 Résumé

Nous avons vu dans ce chapitre le développement de logiciel avec Mirage soit les structures de données et la bibliothèque de fonctions disponibles pour le développeur. Dans le prochain chapitre, nous présentons une description détaillée de Mirage.

Chapitre 3. Description détaillée de Mirage

Dans le chapitre précédent, il était question de développement de logiciel avec Mirage. Dans le présent chapitre, la structure interne de Mirage est décrite en détail.

3.1 La structure de Mirage

La structure globale de Mirage est représentée dans le diagramme de la Figure 3.1.

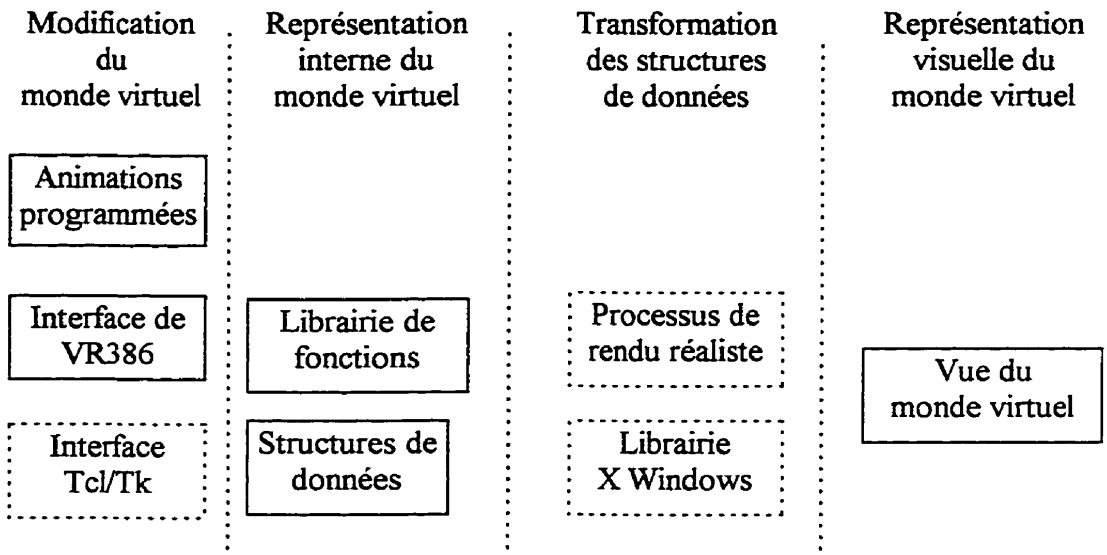


Figure 3.1 Structure globale de Mirage

Les modules développés dans le cadre de ce projet sont encadrés en pointillé et sont traités dans ce chapitre:

- le processus de rendu réaliste [*rendering*];
- la bibliothèque X Windows, utilisée pour l'affichage visuel et la gestion du clavier et de la souris;
- l'interface usager Tcl/Tk, qui reproduit toutes les fonctions de l'interface usager originale de VR386.

La structure de Mirage est constituée en résumé d'une boucle continue, qui gère les divers événements provenant du clavier et de la souris, et qui régénère une vue du monde virtuel en fonction de ces événements et des diverses animations programmées. La structure du module principal est la suivante:

```
void main(int argc, char *argv[])
{
  Initialisations(argc,argv);  // Initialise
  MainLoop();                  // Boucle de Mirage
}
```

Figure 3.2 Fonction `main()` de Mirage tirée de `Xmain.C`

Mirage effectue donc diverses initialisations (ouverture d'une fenêtre X, chargement optionnel d'un fichier `WLD`, lecture des menus Tk, etc...) et entre dans une boucle

continue. Cette boucle est interrompue uniquement à la sortie du programme. Le fichier `Xmain.C` est donné en entier en annexe G.

Vue sous forme différente, la structure de Mirage est illustrée par le diagramme de la Figure 3.3.

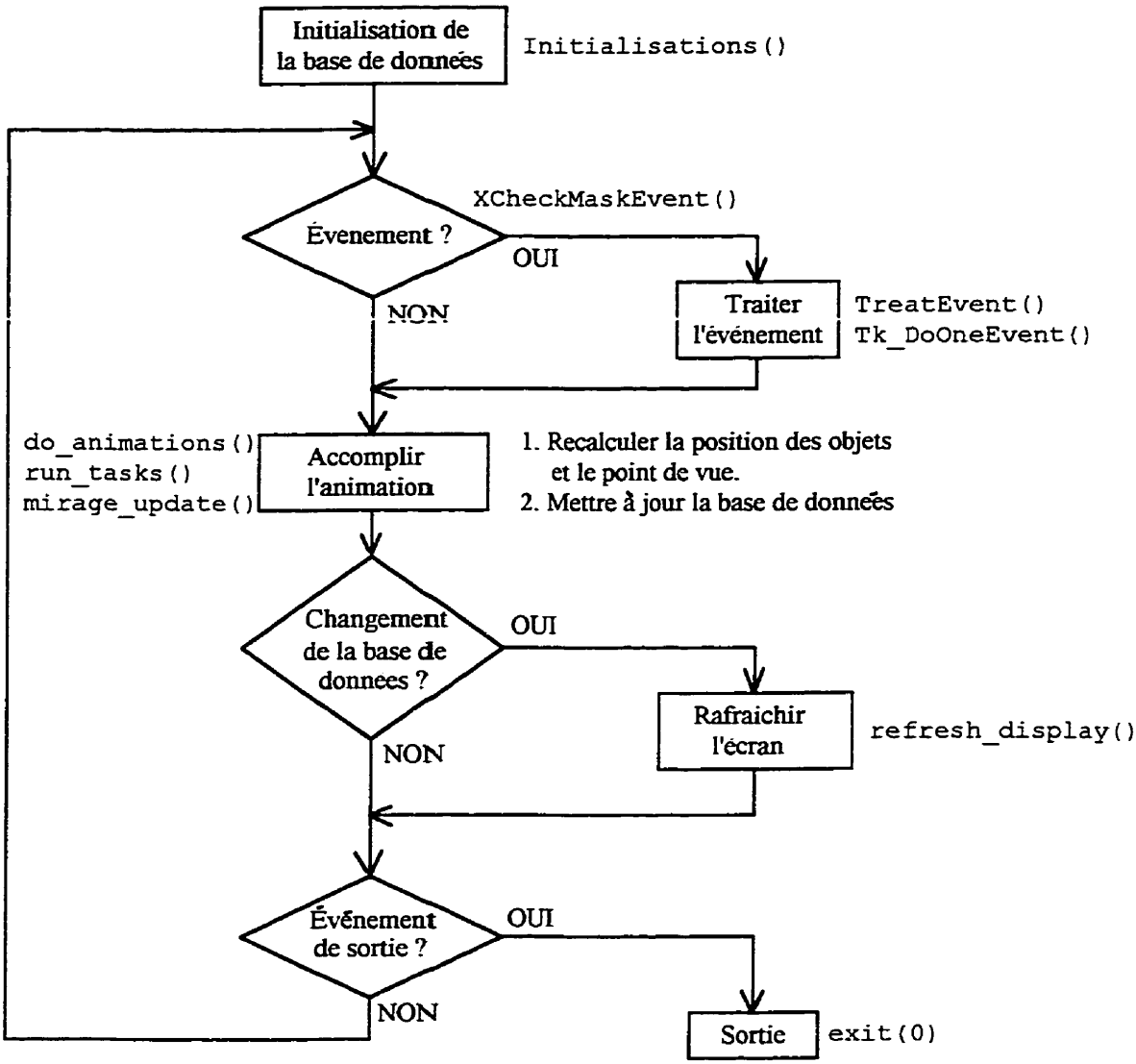


Figure 3.3 Organigramme global de Mirage

3.2 Le processus de rendu réaliste [*rendering*]

Afin de visualiser le monde virtuel, plusieurs étapes sont nécessaires avant d'afficher les objets 3D à l'écran de façon optimale

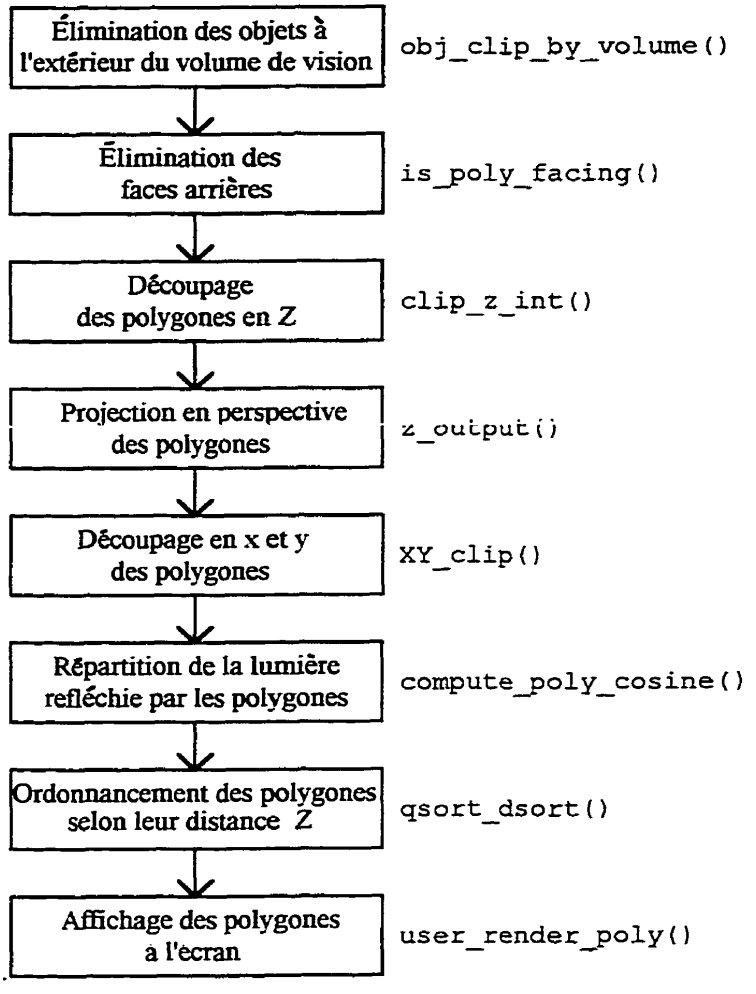


Figure 3.4 Étapes dans l'affichage des polygones (fonction `subrender()`)

Comme ce processus est très classique et donc traité dans plusieurs ouvrages [Watt 1992], on présente les étapes sous forme résumée.

3.2.1 Distances de découpage [*clipping hither* et *yon*]

Les plans avant et arrière ainsi que la fenêtre de visualisation [*hither* et *yon*] définissent un parallélépipède de vision pour les objets. Les objets sont visibles s'ils sont à l'intérieur du parallélépipède de vision, et invisibles dans le cas contraire. Une prédéformation des coordonnées transforme la pyramide tronquée de la Figure 3.5 en un parallélépipède.

Cette étape élimine beaucoup d'objets, qui ne seront pas transférés aux étapes subséquentes.

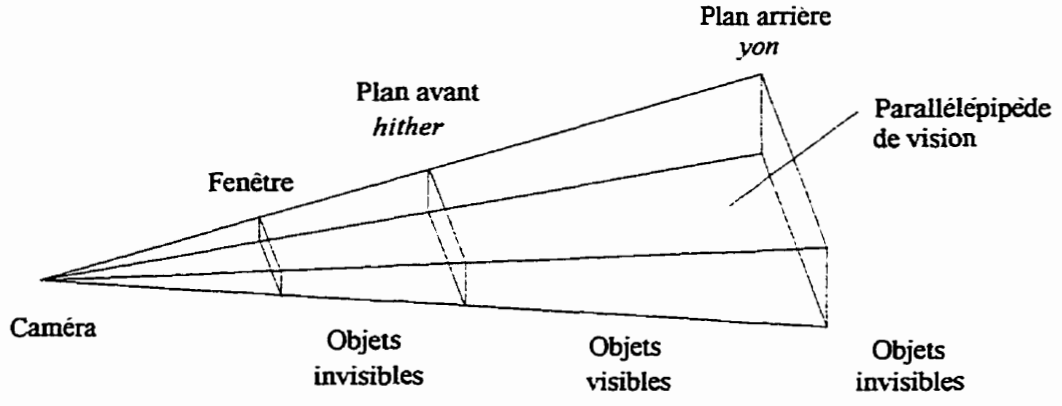


Figure 3.5 Parallélépipède de vision

3.2.2 Élimination des faces arrières [*backface culling*]

Les polygones dans Mirage ne sont visibles que d'un seul côté. Les polygones qui ne font pas face à la caméra ne sont donc pas considérés lors de l'affichage. L'élimination se fait par un test de la normale du polygone. Si la normale ne fait pas face à la caméra, le polygone est rejeté.

L'ordre des points lors de la définition du polygone est donc important car il détermine son orientation. Les polygones invisibles ne seront pas traités ultérieurement.

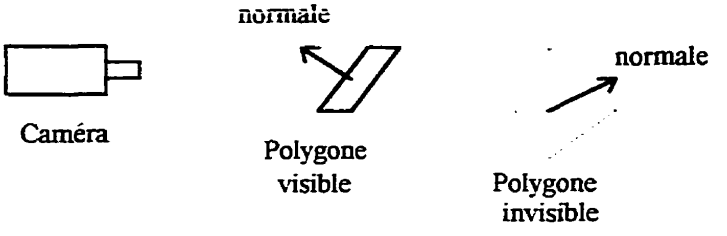


Figure 3.6 Test de la normale des polygones

3.2.3 Découpage des polygones en x et y

Une fois que les coordonnées des polygones ont été calculées, les polygones qui ne sont pas entièrement compris dans le parallélépipède de vision doivent être découpés (ou détourés) avant d'être affichés à l'écran. Les segments coupant un des plans du parallélépipède de vision sont interceptés avec celui-ci pour créer un nouveau sommet du

polygone. La coordonnée Z de ce point est également calculée, puis les polygones sont ordonnés suivant leur profondeur. L'algorithme de Sutherland-Hodgman [Sutherland, Hodgman, 1974] est utilisé pour cette tâche. Le découpage est implanté de manière récursive. Le polygone est testé avec un côté du rectangle et découpé si nécessaire, puis la fonction de découpage est rappelée avec le nouveau polygone ainsi créé.

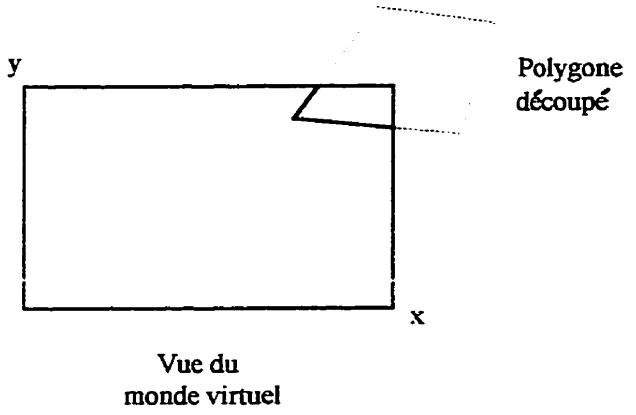


Figure 3.7 Découpage en x et y des polygones

3.2.4 Calcul des intensités de lumière réfléchies [*shading*]

Pour les polygones possédant une couleur prédéfinie, aucun calcul n'est nécessaire. Certains polygones ont une couleur qui varie selon l'orientation de la normale du polygone et la position ou direction des sources de lumière. Pour ces polygones, 16 intensités sont possibles. Mirage utilise le modèle de Lambert. Chaque polygone est colorié avec une intensité constante. L'intensité dépend alors du produit scalaire du vecteur normal du polygone et du vecteur donnant la direction de la lumière:

$$I = k_d \cdot I_a + k_d \cdot I_p \cos\theta$$

où k_d est la réflectivité du matériau, I_a est la fraction de lumière absorbée par la surface, et I_p est l'intensité de la source ponctuelle. De plus on a: $\cos\theta = (N \bullet L)$ où N est la normale à la surface et L est le vecteur de lumière illustré dans la figure suivante:

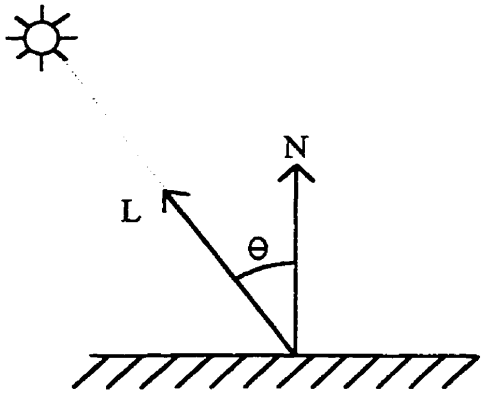


Figure 3.8 Calcul de réflexion diffuse d'un matériau

Dans Mirage, aucun facteur d'atténuation de la lumière en fonction de la distance n'est utilisé.

3.2.5 Algorithme du peintre

Le noyau du processus de rendu réaliste [*rendering*] est l'algorithme du peintre. D'abord la profondeur en Z de chaque polygone est calculée, puis les polygones sont ordonnés dans le sens décroissant de profondeur. Les polygones sont donc affichés par ordre de profondeur décroissant. Les polygones les plus éloignés sont tracés en premier. Suivant le nombre de polygones à ordonner, il est préférable d'utiliser un algorithme de tri rapide [*quicksort*] pour un grand nombre de polygones ou un algorithme de tri par insertion pour un nombre réduit de polygones. L'algorithme utilisé par Mirage ne tient pas compte

des recouvrements de polygones et génère donc parfois des anomalies dans les priorités attribuées aux polygones (apparition de polygones cachés).

3.2.6 Texture

La texture ne fait pas partie de VR386, mais a été rajoutée dans Mirage. Une texture est une image de trame [*bitmap*] qui est collée sur un polygone. L'image est généralement stockée sur disque dur sous la forme d'un fichier de type PCX. L'utilisation de textures résulte en un modèle plus réaliste du monde réel. Le désavantage de son utilisation est la diminution de la performance due à la lenteur de construire à l'écran un polygone texturé par rapport à un polygone d'une couleur unique.

3.2.7 Palette de couleurs de Mirage

Mirage a une palette de 256 couleurs. Celle-ci est prévue au départ pour les cartes graphiques standard sur les postes de travail Sun et ordinateurs personnels fonctionnant sur Linux. Les seize premières couleurs contiennent les seize couleurs de base standard sur un PC. Le reste des couleurs est utilisé pour la répartition de la lumière [*shading*]. L'intensité dépendra de la position ou direction de la lumière et de l'orientation du polygone.

3.3 Les fonctions X Windows de Mirage

Les routines graphiques de Mirage se basent sur les fonctions graphiques X Windows. Les fonctions de cette bibliothèque `Xlib` envoient des requêtes au serveur X [Recanati 1993]. Il est donc important d'expliquer cette partie de Mirage.

3.3.1 Les événements dans X Windows

Le système X Windows fonctionne par événements. La structure d'un logiciel développé avec X Windows est généralement une boucle continue, qui traite des événements venant du clavier ou de la souris. La structure de Mirage est de cette forme. La liste des événements traités par Mirage est donnée dans le Tableau 3.1.

Tableau 3.1 Les événements X Windows traités par Mirage

Événement	Description	Action
`KeyPress`	Appui d'une touche au clavier	Appel des menus suivant la touche
`Expose`	La fenêtre est mise à découvert	Mise à jour de la fenêtre
`ButtonPress`	Appui d'un bouton de la souris	Déplacement ou sélection d'un objet
`ButtonRelease`	Un bouton de la souris est relâché	Arrêt du déplacement du point de vue
`MotionNotify`	Déplacement de la souris	Déplacement si un des boutons enfoncé
`ConfigureNotify`	Dimension de fenêtre changée	Mise à jour de la fenêtre
`EnterNotify`	Curseur entre dans la fenêtre	Changement de la palette de couleur

3.3.2 La gestion des événements X Windows

Les événements X Windows sont gérés par des fonctions gestionnaires [appelées *handlers*]. A chaque événement correspond une fonction gérant l'événement. Par exemple, la fonction `TreatKeyPressEvent` traite l'événement `KeyPress`.

3.3.3 Les principales fonctions X Windows de Mirage

Mirage utilise les fonctions X Windows afin d'afficher des polygones et du texte dans sa fenêtre.

3.3.4 L'utilisation de pixmaps

L'animation dans VR386 utilisait 3 pages graphiques de la carte VGA. La technique de triple-tamponnage [*triple-buffering*] est utilisée pour créer des animations fluides à l'écran. Le dessin de polygones ou de texte se fait sur une page graphique qui n'est pas affichée à l'écran. Une fois le processus de dessin terminé, la page est affichée à l'écran. Ces pages sont stockées dans la mémoire de trame de la carte vidéo.

Dans X Windows, les pages graphiques sont remplacées par des pages virtuelles appelées *pixmaps* ou pseudo mémoire de trame. Les graphiques ne sont pas affichés directement dans la fenêtre, mais sont écrits dans une page virtuelle. Une fois que l'image est complétée, elle est recopiée à l'écran. Lors d'une occultation partielle ou complète de la fenêtre, une partie du contenu de la fenêtre est perdue. Le rafraîchissement de la fenêtre

est effectué simplement en recopiant le contenu de la page virtuelle dans la fenêtre. Les *pixmaps* sont définis ainsi:

```
Pixmap pixmap[3];
```

Les *pixmaps* ont la même dimension que la fenêtre de Mirage. Lorsque la fenêtre est redimensionnée, les *pixmaps* doivent l'être également.

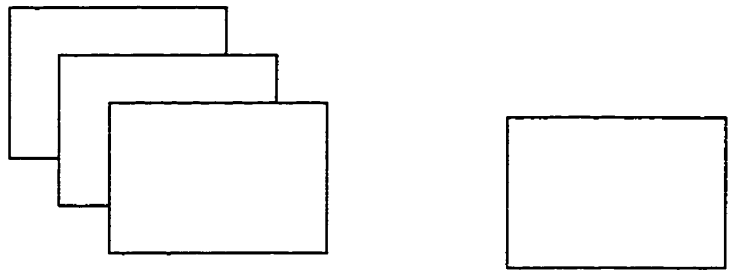


Figure 3.9 Trois *pixmaps* et la fenêtre principale de Mirage

3.4 L'interface Tcl/Tk

3.4.1 Le langage Tcl et la bibliothèque Tk

Tcl (prononcé « tickle ») est un langage interprété développé par John K. Ousterhout, professeur du département de génie électrique et informatique de l'Université de Berkeley en Californie [Ousterhout 1994]. Ce langage est développé entièrement en langage C. Tk est une bibliothèque de fonctions de haut niveau servant à créer des interfaces usager sous X Windows, et a été développée à partir de la bibliothèque Tcl.

Tcl/Tk est d'utilisation facile et très puissant. Le code source et les bibliothèques sont disponibles gratuitement sur l'internet au site web de Sun Labs:

[http://www.sunlabs.com/research/tcl/install.html].

En utilisant Tk, le développement d'interfaces pour l'environnement X Windows est facilité. Les bibliothèques Tcl et Tk peuvent être combinées à un logiciel écrit en langage C pour y rajouter une interface graphique conviviale. En raison de conflits dans la gestion d'événements X Windows, il est beaucoup plus simple de créer une fenêtre séparée pour les menus construits à l'aide de Tcl/Tk, au lieu d'utiliser une fenêtre partagée. Les menus de la fenêtre Tcl/Tk contrôleront notre application graphique développée en langage C ou C++ dans la fenêtre principale. Mirage utilise les versions suivantes: Tcl 7.6 et Tk 4.2.

3.4.2 L'interface entre Tcl/Tk et une application développée en langage C

Le language Tcl/Tk est un langage interprété durant l'exécution du logiciel Mirage. Mirage contient donc un interpréteur Tcl/Tk intégré. Comme Tcl/Tk utilise ses propres variables, une communication entre l'interpréteur Tcl/Tk et le logiciel écrit en langage C est nécessaire. Les détails de cette communication sont donnés ici afin que le développeur puisse rajouter d'autres fonctionnalités aux menus de Mirage.

3.4.2.1 Initialisations

Pour utiliser Tcl/Tk à l'intérieur d'un logiciel écrit en langage C, il faut inclure les lignes suivantes au début du programme:

```
#include <tcl.h>
#include <tk.h>
```

Il faut également rajouter les options de compilation `-tcl7.6 -tk4.2` qui indiqueront au compilateur d'inclure les bibliothèques Tcl (`libtcl7.6.so`) et Tk (`libtk4.2.so`) dans l'édition des liens. Dans le fichier `makefile`, ces options se rajoutent aux options déjà présentes.

A l'intérieur du programme C, il faut tout d'abord créer un interpréteur Tcl avec la commande `Tcl_CreateInterp()`. La fenêtre Tk peut ensuite être créée avec un appel de la fonction `Tk_Init()`. Les scripts Tcl/Tk de Mirage sont contenus dans des

fichiers d'extension `tk`. Ces scripts sont évalués avec la fonction `Tcl_EvalFile()`. Ainsi le fichier `menu.tk` est évalué dès le lancement de Mirage.

3.4.2.2 Gestion des événements de la fenêtre Tk

Dans la boucle principale de Mirage, un appel à `Tk_DoEvent()` s'assure que les événements de la fenêtre Tcl/Tk sont traités. Ces événements sont générés par le déclenchement d'une gachette à l'aide d'un péripherique d'entrée, entre autres un appui sur une touche du clavier, sélection de menu avec la souris, etc...

3.4.2.3 Communication entre Tcl et C

Il y a plusieurs façons de communiquer entre ces deux langages. Une première méthode utilisée est de créer un lien entre deux variables par exemple, avec la commande:

```
Tcl_LinkVar(interp,"wireframe",(char *)&wireframe,
TCL_LINK_INT);
```

Un lien est créé entre la variable `wireframe` de Tcl et `wireframe` en C. Ainsi, une modification de la variable Tcl entraîne la même modification dans la variable en langage C. Les deux variables doivent être du même type. Ici, les variables sont de type `int`.

L'autre méthode de communication est de récupérer ou modifier une variable Tcl à partir du C avec un appel à `Tcl_GetVar()` ou `Tcl_SetVar()` respectivement.

3.4.3 Les menus de Mirage

La description de la console Tk de Mirage est contenue dans le script `menu.tk`. Au début du fichier se trouve la description générale de la fenêtre, soit les menus accessibles dans la fenêtre, puis une zone de texte utilisée pour afficher des messages.

```
frame .mbar -relief raised -bd 2
frame .dummy -width 17c -height 0c
pack .mbar .dummy -side top -fill x
menubutton .mbar.file -text File -underline 0 -menu
  .mbar.file.menu
menubutton .mbar.display -text Display -underline 0 -menu
  .mbar.display.menu
menubutton .mbar.view -text View -underline 0 -menu
  .mbar.view.menu
menubutton .mbar.object -text Object -underline 0 -menu
  .mbar.object.menu
menubutton .mbar.figure -text Figure -underline 0 -menu
  .mbar.figure.menu
menubutton .mbar.demo -text Demo -underline 0 -menu
  .mbar.demo.menu
menubutton .mbar.help -text Help -underline 0 -menu
  .mbar.help.menu
pack .mbar.file .mbar.display .mbar.view .mbar.object
  .mbar.figure .mbar.demo -side left
pack .mbar.help -side right
text .text -relief raised -bd 2 -yscrollcommand ".scroll
  set"
scrollbar .scroll -command ".text yview"
pack .scroll -side right -fill y
```

Figure 3.10 Description de la fenêtre Tk dans le fichier menu.tk

Il y a principalement quatre catégories de commandes dans le menu.

3.4.3.1 Le bouton de type checkbutton

Une variable est associée à ce type de bouton avec la commande `-variable`. La variable est activée ou désactivée avec ce bouton. Dans Mirage, ces variables Tcl sont associées à des variables C. Les variables C sont donc modifiées directement par ces boutons.

Ex: La variable `wireframe` est associée à l'option `Wireframe` du menu `display`. La première lettre de `Wireframe` est soulignée (index 0), et la touche de raccourci « `Ctrl+w` » est affichée.

```
.mbar.display.menu add checkbutton  -label "Wireframe" -
 underline 0 -accelerator "Ctrl+w" -variable wireframe
```

Figure 3.11 Exemple de bouton checkbutton

3.4.3.2 Le bouton de type radiobutton

Ce bouton sert à donner un choix entre plusieurs valeurs pour une variable donnée. La variable est nommée avec l'option `-variable` et sa valeur par l'option `-value`.

Ex: La variable `light` peut prendre la valeur `s` ou `p` suivant la sélection de `Spotlight` ou `Pointlight`, correspondant à une lumière d'appoint ou une source de lumière ponctuelle.

```
.mbar.display.menu add radiobutton -label "Spotlight" -
 variable light -value s
.mbar.display.menu add radiobutton -label "Pointlight" -
 variable light -value p
```

Figure 3.12 Exemple de bouton radiobutton

3.4.3.3 Le bouton de type cascade

Ce type de bouton fait appel à un sous-menu. Il est utilisé pour créer des menus en cascade. Les sous-menus sont décrits de la même manière que les menus précédents.

Ex: Le menu appelé `Movement step` fait appel au sous-menu `.mbar.view.menu.mstep`.

```
.mbar.view.menu add cascade -label "Movement step" -menu
  .mbar.view.menu.mstep

menu .mbar.view.menu.mstep
.mbar.view.menu.mstep add radiobutton -label "20" -variable
 mstep -value 20
.mbar.view.menu.mstep add radiobutton -label "50" -variable
 mstep -value 50
...
```

Figure 3.13 Exemple de menu en cascade

3.4.3.4 Le bouton de type command

Ce bouton exécute une commande Tcl/Tk donnée par l'option `-command`. Par exemple, dans Mirage, la commande `set cname view_info` peut être appelée. Dans le programme, la valeur de `cname` est lue avec la fonction `Tcl_GetVar`, puis recherchée dans la liste des commandes de Mirage (un tableau de chaînes de caractères). Si une correspondance est obtenue, la fonction de rappel appropriée est appelée.

Ex: L'option `Load` du menu `Object` est associée à la commande Tcl `set cname object_load`.

```
.mbar.object.menu add command -label "Load" -underline 0 -
 accelerator "Ctrl+l" -command "set cname object_load"
```

Figure 3.14 Exemple de bouton command

Dans le programme C, la variable Tcl cname est lue, puis recherchée dans la liste de commandes:

```
char *cname;
// On associe à cname la valeur de « cname » de Tcl
cname = Tcl_GetVar(interp,"cname",0);

// on vérifie si cname contient une valeur
if (strlen(cname) > 2)
  { // on parcoure le tableau command_list
    for (i=0;command_list[i] != NULL; i++)
      // comparer avec la valeur de cname
if (!strcmp(command_list[i],cname))
 break;

//Selon la valeur de i, la fonction appropriée est appelée:

switch (i)
{
    .
    case c_object_load: object_load(); break;
    case c_object_save: object_save(); break;
    .
 }
```

Figure 3.15 Une partie du fichier tkcommands.C

3.4.4 Les menus déroulants [*pop-up*] de Mirage

Afin d'afficher certaines informations, ou demander une valeur à l'usager, Mirage utilise des menus déroulants [*pop-up*]. Ces menus sont également développés à l'aide de Tcl/Tk.

Les scripts sont contenus dans les fichiers `dialog.tk` et `input.tk`. Le script de `dialog.tk` sert à afficher des informations dans une fenêtre, tandis que le script d'`input.tk` demande une information à l'usager. Le développement de ces menus est un peu plus complexe que celui de la console de Mirage.

3.5 Élimination des modules écrits en assembleur

Une partie importante des modules de VR386, à l'origine, ont été développés en assembleur. Ces modules sont associés au processus de rendu réaliste. Comme le processus de rendu réaliste est extrêmement critique du point de vue de la performance d'un logiciel RV, dans VR386 il a été développé en langage assembleur.

Les modules écrits en assembleur ont été réécrits en langage C pour Mirage. Afin d'éliminer les erreurs dès les premières étapes de développement de Mirage, ces modules ont été intégrés et testés individuellement. Les erreurs pouvaient donc être localisées dans des fichiers individuels et donc être éliminées plus facilement.

Après avoir traduit tous les fichiers écrits en assembleur, nous avons obtenu une version de VR386 entièrement en langage C à l'exception des pilotes DOS. Les pilotes DOS et fonctions associées à la gestion de l'écran, du clavier et de la souris ont été remplacés par des fonctions X Windows. Les fonctions associées à d'autres périphériques tel que la manette de jeux, le gant [powerglove] et la souris 6D ont été simplement éliminées.

3.6 Résultats

La performance de Mirage est donnée ci-dessous, évaluée sur un Pentium cadencé à 60 Mhz. La Figure 3.16 montre le nombre d'images générées par seconde en fonction du nombre total de polygones dans le monde virtuel. Ces données ont été recueillies en visualisant des sphères générées par le programme `sphere` donné en annexe.

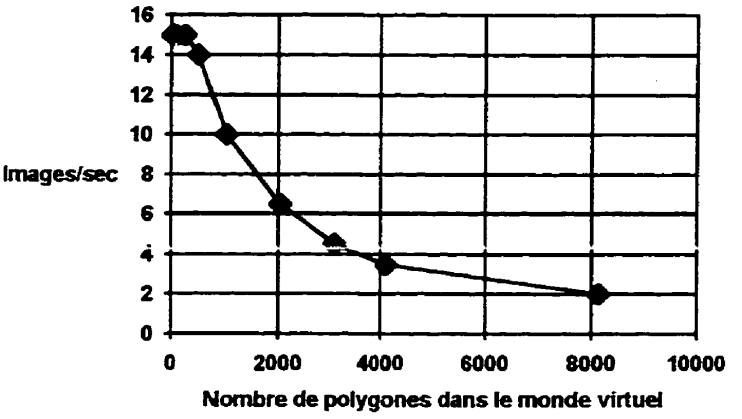


Figure 3.16 Nombre d'images par seconde en fonction du nombre total de polygones

La forme de cette courbe s'explique de la façon suivante. Le temps requis pour dessiner n polygones peut être donné par l'équation:

$$t_n = t_c + t_p.n$$

où

t_c est un temps constant requis pour effacer le tampon correspondant à la pseudo-mémoire de trame [*pixmap*], le recopier dans la fenêtre X et d'autres opérations n'impliquant pas directement les polygones.

t_p est le temps requis pour dessiner un seul polygone.

Le nombre d'images affichés par seconde (correspondant à la Figure 3.16) pour un monde virtuel à n polygones est donc:

$$f_n = \frac{1}{t_c + t_p.n}$$

Le nombre de polygones affichés par seconde (correspondant à la Figure 3.17) est:

$$f_p = \frac{n}{t_c + t_p.n} = \frac{1}{\frac{t_c}{n} + t_p}$$

La Figure 3.17 montre le nombre de polygones générés par seconde en fonction du nombre total de polygones dans le monde virtuel. Cette valeur plafonne à environ 14 300 polygones/sec, une valeur qui donne une bonne indication de la performance du contrôleur graphique utilisé.

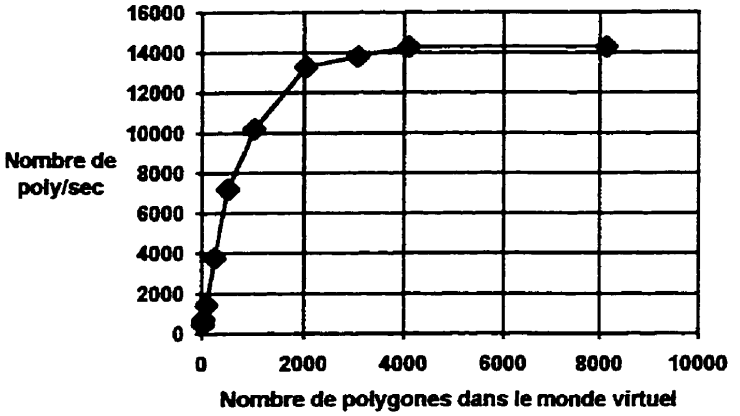


Figure 3.17 Nombre de polygones affichés par seconde en fonction du nombre total de polygones

3.7 Résumé

Dans ce chapitre, la description de la structure interne de Mirage a été donnée. Nous avons détaillé le processus de rendu réaliste, l'utilisation de la bibliothèque X Windows, l'interface entre le langage C et les bibliothèques Tcl/Tk et les résultats obtenus. Le chapitre suivant traite des améliorations qui pourraient être apportées à Mirage et des conclusions qui ont été tirées de ce projet.

Conclusions

Dans ce chapitre, nous présentons les caractéristiques et avantages principaux de Mirage, ainsi que certains problèmes non-résolus. Quelques directives quant à l'évolution et les développements futurs de Mirage sont élaborées.

4.1 Caractéristiques principales de Mirage

Ce mémoire présente un logiciel de développement de logiciels de réalité virtuelle nommé Mirage. Les caractéristiques de Mirage sont les suivantes:

- logiciel multiplate-forme (PC sous Linux, Sun, SGI, ...) se servant de la bibliothèque X Windows;
- code source hautement modularisé en langage C et entièrement accessible;
- gratuit à l'inverse d'autres logiciels similaires;
- bibliothèque de fonctions de haut niveau pour manipuler des objets et hiérarchies d'objets virtuels;
- création rapide de mondes virtuels sous forme de fichiers de type `WLD` se servant d'objets pré-existants disponibles sur l'internet;
- format `PLG` est un standard connu;
- formats de fichiers plus simples que VRML 1.0 ou 2.0;

- possibilité de convertir les mondes virtuels de Mirage (fichiers `WLD`) au format VRML 1.0;

- interface graphique construite avec les bibliothèques Tcl/Tk facilement modifiable.

De nombreux compromis ont été nécessaires lors du développement de Mirage. Puisque OpenGL n'était pas supporté directement sur toutes les plates-formes UNIX au début de ce travail de recherche (début 1995), ce langage n'a pu être utilisé pour développer Mirage. Les avantages d'OpenGL n'ont donc pas pu être exploités. De plus, la simplicité de l'algorithme de rendu réaliste peut produire quelques erreurs mais résulte en une génération plus rapide de la scène (plus grand nombre d'images/seconde). Le sentiment d'immersion dans un monde virtuel est alors accru.

4.2 Problèmes demeurant dans Mirage

Quelque problèmes mineurs subsistent dans Mirage:

- Problèmes avec les subdivisions de polygones, entraînant la visibilité de certains polygones. Les objets partiellement superposés sont aussi souvent affichés incorrectement. Ces problèmes sont dus à la méthode utilisée dans le rendu réaliste;

- Les variables d'état déclarées à l'intérieur de scénarios d'animation dans les fichiers de type `WLD` ne fonctionnent pas.

La manière dont Mirage effectue le rendu réaliste des objets à l'écran peut souvent créer des erreurs de visibilité entre les objets. Un polygone peut paraître devant un autre alors

qu'en réalité il se trouve derrière. L'algorithme de rendu réaliste ne tient pas compte des nombreuses erreurs qui peuvent se produire. Le traitement de ces erreurs ralentirait quelque peu Mirage tout en donnant un rendu réaliste plus exact des objets. Les auteurs de Rend386 ont choisi de ne pas traiter ces erreurs.

Le problème est partiellement résolu en utilisant des subdivisions de polygones [*splits*], qui sont des plans imaginaires séparant les objets dans une scène. En plaçant plusieurs plans de ce genre dans une scène, on organise les objets dans une structure hiérarchique d'arbre binaire. Ces plans donnent des priorités aux objets dans l'algorithme de rendu réaliste, réduisant ainsi le nombre d'erreurs de visibilité. L'inconvénient majeur des subdivisions de polygones [*splits*] est que les plans imaginaires doivent être placés aux bons endroits par l'usager dans le fichier `WLD`, ce qui n'est pas facile. L'utilisation de ces plans imaginaires n'est donc qu'une solution temporaire. Pour résoudre ces problèmes, l'algorithme de rendu devrait être remplacé par un algorithme plus fiable (utilisant l'algorithme du *z-maximal [z-buffer]* à l'aide de fonctions d'OpenGL par exemple).

Une autre lacune de Mirage est le contrôle limité qu'offre la bibliothèque sur le monde virtuel. Il est difficile d'éliminer ou rajouter des objets à volonté pendant l'exécution du programme. La construction de grands environnements tels que des immeubles contenant plusieurs salles est donc difficile. A l'entrée d'une pièce, il faudrait être capable de charger les objets contenus à l'intérieur, puis de s'en débarrasser lorsqu'on quitte la salle. Cette possibilité n'est pas intégrée à l'intérieur de Mirage. Les mondes virtuels multiples ne sont également pas pris en compte.

4.3 Améliorations pouvant être apportées

Mirage utilise les tampons de pseudo mémoire de trame [*pixmaps*] de X Windows pour accomplir la technique de triple tamponnage [*triple-buffering*] pour obtenir des animations fluides. Il existe une extension à X Windows spécialement conçue pour le double tamponnage [*double-buffering*] et le triple tamponnage [*triple-buffering*]. Utilisée correctement, cette extension donnerait éventuellement de meilleurs résultats que l'utilisation des tampons de mémoire de trame [*pixmaps*].

La mémoire partagée n'est pas utilisée dans Mirage. Son utilisation pourrait également bien améliorer les performances au niveau graphique.

4.4 Le processus de rendu réaliste [*rendering*]

Le rendu réaliste [*rendering*] dans Mirage est effectué par un engin rapide qui utilise l'algorithme du peintre. Les tracés graphiques sont effectués par des routines X Windows. Il n'y a aucune fonction 3D dans la bibliothèque X Windows. Ces fonctions doivent donc être fournies par le développeur. Les fonctions de remplissage de polygones Gouraud [Gouraud 1971] prenant en compte les sources de lumière ou les fonctions d'affichage des textures doivent être fournies par l'usager. Il apparaît donc nécessaire de miser plutôt sur une bibliothèque graphique 3D dans laquelle ces fonctions sont implantées; la bibliothèque OpenGL disponible sur de nombreuses plates-formes nous apparaît un bon choix. La grande majorité des fonctions 3D sont déjà implantées et documentées. Un logiciel de réalité virtuelle basé sur OpenGL sera probablement moins

performant qu'un logiciel ayant ses propres fonctions 3D optimisées mais cela changera avec les nouvelles cartes graphiques optimisées pour OpenGL. Il faut noter que des versions d'OpenGL existent pour Linux, Windows 95, NT, OS/2, Solaris, etc. La bibliothèque graphique Mesa 3D implante environ 90% des fonctions d'OpenGL. Cette bibliothèque est disponible gratuitement et existe sur les plate-formes Unix et Windows. Mirage GL serait donc portable à de nombreux postes de travail. Un étudiant a déjà porté Mirage à OpenGL lors d'un projet de fin d'études [Tantely 1996] cependant ce prototype n'offre qu'une petite partie des fonctionnalités de Mirage.

4.5 La bibliothèque de fonctions

Une bonne bibliothèque de fonctions est essentielle dans un logiciel de réalité virtuelle. Une telle bibliothèque comporte des fonctions pour manipuler les objets, modifier leurs propriétés et contrôler le cours d'une simulation. Une bonne bibliothèque orientée-objet en langage C++ posséderait de nombreux avantages sur la bibliothèque actuelle de Mirage. Le développement d'applications s'en trouverait facilité. Par exemple, les vecteurs 3D seraient plus facilement manipulés grâce à l'utilisation de la surcharge d'opérateur.

4.6 Le format VRML

Mirage utilise des métafichiers graphiques pour décrire les mondes virtuels; les formats `PLG`, `NFF` et `DXF` sont présentement supportés. Le format interne utilisé par Mirage est le format `PLG`; les fichiers aux formats `NFF` et `DXF` sont préalablement convertis au

format PLG. Il serait utile de supporter directement le format VRML 1.0 ou 2.0 (*Virtual Reality Modelling Language*) étant donné sa popularité grandissante et sa définition précise. Le code source pour lire ces fichiers est disponible gratuitement sur l'internet dans la bibliothèque qvlib pour VRML 1.0. Ce n'est donc pas un énorme effort pour construire un logiciel de visualisation de mondes VRML 1.0 si nous possédons déjà un engin graphique, cependant la tâche est beaucoup plus ardue pour le langage VRML 2.0.

4.7 L'interface usager

L'interface est très importante pour tout logiciel. Mirage possède une interface utilisant la bibliothèque Tcl/Tk. Ce langage graphique est beaucoup plus facile à utiliser que la bibliothèque Motif et a l'avantage d'offrir les mêmes fonctionnalités en plus d'être gratuite. La documentation sur Tcl/Tk laisse à désirer et ne montre pas très bien comment utiliser ce langage avec une application écrite en langage C mais la documentation sur Mirage remédie à cette lacune. Tcl offre la possibilité de rajouter un langage interprété à un logiciel de réalité virtuelle et donc une plus grande interactivité.

La bibliothèque Motif offre également toutes les fonctions nécessaires à la construction d'interfaces usager. Son apprentissage est plus difficile que Tcl/Tk et cette bibliothèque n'est pas gratuite, mais elle possède cependant plusieurs avantages. Motif s'intègre mieux à une application graphique X Windows que Tcl/Tk et son utilisation est beaucoup plus répandue.

Si on décide d'utiliser OpenGL, la bibliothèque GLUT (*Graphics Library Utility Toolkit*) [Kilgard 1996] permet de créer des menus et de simplifier la création d'applications OpenGL. La bibliothèque GLUT est disponible gratuitement sur l'internet et ne se limite pas aux plates-formes Unix.

4.8 Les directions possibles

Il semble utile d'améliorer Mirage. Ces améliorations touchent essentiellement à l'engin graphique, à la bibliothèque de Mirage et au support du format VRML. Les possibilités sont les suivantes:

- un engin graphique basé directement ou indirectement sur la bibliothèque OpenGL;
- une bibliothèque de fonctions orientée objet en C++ pour le développement d'applications [*API - Application Programmer Interface*];
- l'intégration à Mirage du support du format VRML;
- une interface usager créée avec Motif ou GLUT;
- une perspective multi-usagers.

BIBLIOGRAPHIE

AHUJA R.S., CLOUÂTRE M. (1996). Two-Handed Teleoperation in Mirage, Design project report, Department of Electrical Engineering, McGill University.

AMES A.L., NADEAU D.R., MORELAND J.L. (1996). The VRML Sourcebook, John Wiley & Sons.

AUGER C. (1997). Simulation de l'assemblage mécanique par la réalité virtuelle. Projet de fin d'études, Dépt. de génie mécanique, École Polytechnique de Montréal.

BAJURA M., FUCHS H., OHBUCHI R. (1992). Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient, Proceedings of SIGGRAPH '92, 203-210.

BELANGER F. (1996). Importation de fichiers 3D vers un environnement virtuel, Projet de fin d'études, Dépt. de génie mécanique, École Polytechnique de Montréal.

BLINN J.F., NEWELL M.E. (Oct. 1976). Texture and Reflection in Computer Generated Images, Comm. ACM, 542-547.

CROW F. C., (1984). Summed-area Tables for Texture Mapping, Computer Graphics, Proceedings of SIGGRAPH '84, 207-212.

D'ANJOU A. (1996). Files Format on Virtual Reality Software, Rapport de stage, Département de génie électrique, École Polytechnique de Montréal.

FOLEY J., VAN DAM A. (1990). Computer Graphics, Principles and Practice Second

Edition, Addison Wesley.

GARANT E., DESBIENS P., DAIGLE A., RIZZI J.-C., OKAPUU-VON VEH A., SHAIKH A., GAUTHIER R., MARCEAU R.J., MALOWANY A.S. (Juin 1995). Three-Dimentional Modelling for a Virtual Reality Operator Training Simulator, Proceedings of the Stockholm PowerTech Conference, Stockholm, 31-36.

GOURAUD H. (1971). Illumination for Computer Generated Pictures, Comm. ACM, 311-317.

GRADECKI J. (1994). The Virtual Reality Programmer's Kit, Wiley, New York.

HODGES L.F. (1992). Tutorial: Time-Multiplexed Stereoscopic Computer Graphics, IEEE Computer Graphics and Applications, Vol. 12, No. 3, 20-30.

KELLEY A., POHL I. (1990). A Book on C, Programming in C, Benjamin/Cummings Publishing Company.

KILGARD M.J. (Nov. 1996). The OpenGL Utility Toolkit (GLUT) Programming Interface, Silicon Graphics Inc.
Internet:
`http://reality.sgi.com/employees/mjk_asd/glut3/glut3.html`

LOFTIN R.B., KENNEY P.J. (Nov.-Déc. 1994). Virtual Environments in Training: NASA's Hubble Space Telescope Mission, Proceedings of the 16th Interservice/Industry Training Systems and Education Conference (I/ITSEC'94), Orlando, FL.

MARK W.R., RANDOLF S.G., FINCH M., VAN VERTH J.M., TAYLOR R.M.

(1996). Adding Force Feedback to Graphic Systems: Issues and Solutions, Proceedings of SIGGRAPH '96, 447-452.

MCKENNA M., ZELTZER D. (1992). Three Dimensional Visual Display Systems for Virtual Environments, Presence: Teleoperators and Virtual Environments, vol 1, No 4, 421-458.

OKAPUU-VON VEH A., MARCEAU R. J., MALOWANY A., DESBIENS P., DAIGLE A., GARANT E., GAUTHIER R., SHAIKH A., RIZZI J. C. (Jan. 1996). Design and Operation of a Virtual Reality Operator-Training System, Proceedings of the 1996 IEEE/PES Winter Meeting, Baltimore, Maryland.

OKOSHI T. (1976). Three Dimensional Imaging Techniques, New York, Academic Press.

OLANO M., COHEN J., MARK M., BISHOP G. (Avril 1995). Combatting Rendering Latency, Proceedings of the 1995 Symposium on Interactive 3D Graphics, 19-24. Internet: `http://www.cs.unc.edu/~cohenj/lowlat.html`

OSTERHOUT J.K. (1994). Tcl and the Tk Toolkit, Addison-Wesley.

NEIDER J., DAVIS T., WOO M. (1993). OpenGL Programming Guide, Addison-Wesley.

RECANATI C. (1993). X Windows, manuel de programmation, Éditions Eyrolles.

ROEHL B. (1994). Playing God, The Waite Group Inc.

ROLFE J.M., STAPLES K.J. (1988). Flight Simulation, Cambridge University Press.

SEGAL M., AKELEY K. (1994). The Design of the OpenGL Graphics Interface, Silicon Graphics Inc.

SHAMANSKY H. (1996). The integration of windowing and 3D graphics. Internet: http://hertz.eng.ohio-state.edu/~hts/opengl/article.html.

SHAW C., GREEN M., LIANG J., SUN Y. (Juillet 1993), Decoupled Simulation in Virtual Reality with the MR Toolkit, ACM Transactions on Information Systems, Volume 11 Number 3, 287-317.

STAMPE D., ROEHL B., EAGAN J. (1993). Virtual Reality Creations, Waite Group Press.

STRAUSS P.S., CAREY R. (1992). An Object-Oriented 3D Graphics Toolkit, Proceedings of SIGGRAPH '92, 341-349.

STURMAN D.J., ZELTZER D. (1993). A Survey of Glove-Based Input, IEEE Computer Graphics and Applications, Vol. 14, No 1, 30-39.

SUTHERLAND I.E., HODGMAN G.W. (1974). Re-entrant Polygon Clipping, Comm. ACM, 32-42.

TAM E.K., MAUREL C., DESBIENS P., MARCEAU R.J., MALOWANY A.S., GRANGER L. (1997). A Low-Cost, PC-Oriented Virtual Environment for Operator Training Simulators, IEEE PICA 97, Mai 11-16, Columbus, Ohio, 358-364. IEEE Transactions on Power Systems.

TANTELY R. (1996). Implantation d'un engin de réalité virtuelle en OpenGL et GLUT API 2.0, Projet de fin d'études, Département de génie informatique, École Polytechnique de Montréal.

WATT A., WATT M. (1992). Advanced Animation and Rendering Techniques, Theory and Pratice, Addison-Wesley.

WELSH M., KAUFMAN, L. (1995). (traduction de COUGNENC R.) Le systeme LINUX, Editions O'Reilly International Thompson, Paris.

WERNECKE J. (1994). The Inventor Mentor: Programming Object-Oriented 3D graphics with Open Inventor, Release 2, Addison-Wesley.

RÉFÉRENCES INTERNET

Vue la nature éphémère des pages Web, les adresses internet données ci-dessous peuvent ne plus correspondre à des adresses valides. Ces pages ont été accédées et vérifiées en avril 1997.

Avril:
`http://sunee.uwaterloo.ca/~broehl/avril.html`
Ceci est le site officiel d'AVRIL. Une brève description est offerte et le code source peut y être téléchargé.

Convertisseurs de formats:
`ftp://sunee.uwaterloo.ca/pub/rend386/converters`
Ce site ftp contient un bon nombre de convertisseurs au format `PLG`, entre autre des formats 3D Studio et `DXF`. Le convertisseur de format `WLD` à VRML est également téléchargeable.

GLUT:
`http://reality.sgi.com/employees/mjk_asd/glut3/glut3.html`
Cette page est le site officiel de la bibliothèque GLUT pour OpenGL et Mesa. On y trouve le code source pour Windows et UNIX, ainsi que la documentation complète en format postscript et html.

Mesa:
`http://www.ssec.wisc.edu/~brianp/Mesa.html`
Ce site officiel de Mesa décrit cette bibliothèque disponible sur plates-formes Windows et UNIX. La bibliothèque peut également y être téléchargée.

Mirage:

http://www.gegi.polymtl.ca/electech/marceau/labrv/labrv.htm

Ce site web à l'École Polytechnique de Montréal contient une description de Mirage ainsi que le code source pour les plates-formes PC sous Linux, Sun et SGI.

MR Toolkit:

http://web.cs.ualberta.ca/~graphics/MRToolkit.html

Ceci est le site officiel de MR Toolkit. Une version peut être téléchargée à condition de remplir un formulaire de license.

OpenGL:

http://hertz.eng.ohio-state.edu/~hts/opengl/article.html.

Cette page contient un article sur OpenGL écrit par Harry Shamanski. Elle contient également des liens à des sites internet et une bibliographie sur OpenGL.

http://www.sgi.com/Technology/OpenGL/index.html

Un index complet contenant des ressources disponibles sur l'internet est donné.

Open Inventor:

http://www.cts.com/~template/Docs/Open_inventor_ds.htm

Ce site décrit les caractéristiques de Open Inventor et les types d'applications pouvant être développées avec cette bibliothèque.

http://www.sgi.com/Technology/Inventor/FAQ.html

Une liste complète des questions les plus demandées (Frequently Asked Questions) sur Open Inventor avec leur réponses.

Powerglove:

http://www.cms.dmu.ac.uk/~cph/pg2.html

Ce site maintenu par Chris Hand décrit le principe de fonctionnement du Powerglove et donne quelques références.

http://www.cms.dmu.ac.uk/~cph/menelli.html
Le circuit du Menelli Box est donné sur ce site. Ce circuit permet de créer une interface entre le Powerglove et le port série d'un ordinateur personnel.

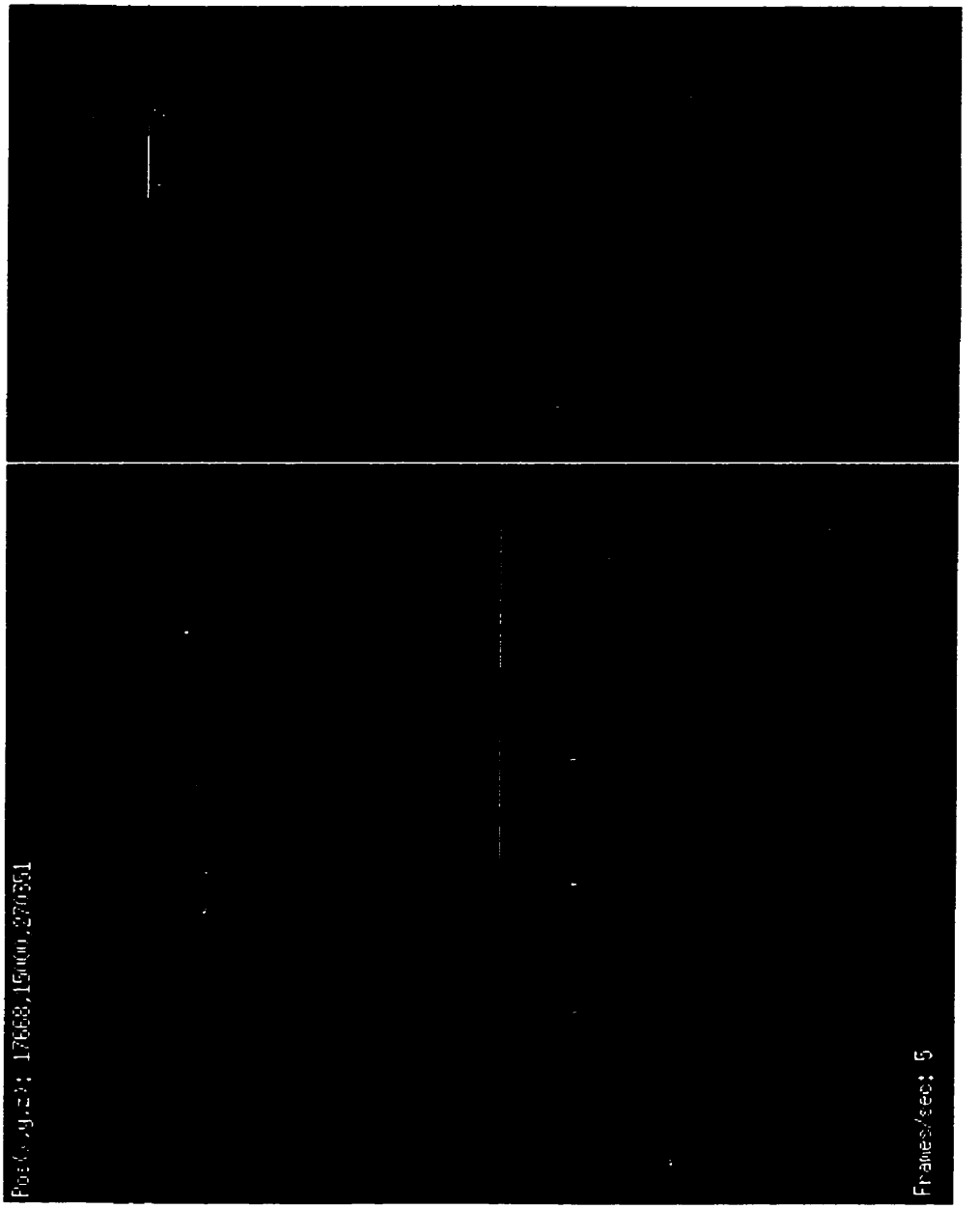
Rend386:
http://www.cms.dmu.ac.uk/~cph/rend386.html
Ce site contient une description de Rend386. Rend386 peut être téléchargé à partir d'un lien ftp donné sur ce site.

UK VR-SIG Object Archive:
http://www.dcs.ed.ac.uk/~mxr/objects.html
Ce site contient un grand nombre d'objets 3D pouvant être téléchargés. Les formats de fichiers présents sont WRL (VRML), BIZ (Division), IV (Open Inventor), PLG (Mirage ou Rend386), RWX (RenderWare) et NFF (WorldToolKit). Les objets peuvent être visionnés avant d'être téléchargés.

VRML:
http://www.sdsc.edu/vrml/browsers.html (VRML Repository)
Cette page web contient une liste exhaustive de logiciels permettant de visualiser des mondes VRML 1.0 et 2.0. De nombreuses plates-formes sont supportées.

WorldToolKit:
http://www.sense8.com/
Ce site officiel de Sense 8 contient des informations sur la compagnie et ses produits, incluant WorldToolKit.

Annexe A Image d'un poste Esope-RV dans Mirage



Annexe B Programme générant une sphère

Paramètres du programme

`sphere <fichier PLG> <rayon> <nb mér.> <nb par.> <code couleur>` où

- <fichier PLG> : nom du fichier PLG qui sera créé
- <rayon> : rayon de la sphère
- <nb mér.>: nombre de méridiens
- <nb.par.>: nombre de parallèles

<code couleur> : code couleur du type Mirage (ex: `0x1EFF`)

Code source de sphere.C

```
// Sphere.C - To compile: g++ -o sphere sphere.C

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct Point {
  long x,y,z;
} Point3d;


Point3d top, bottom;
long nbvertices,nbpolygons;

void write_header(FILE *fp, long nbmer, long nbpar)
{
  nbvertices = 2 + nbpar * (nbmer+1);
  nbpolygons = (nbpar+1) * nbmer;

  fprintf(fp,"Sphere %ld %ld\n",nbvertices,nbpolygons);
}

void write_vertex_to_file(FILE *fp,Point3d vertex)
{
  fprintf(fp,"%ld %ld %ld\n",vertex.x,vertex.y,vertex.z);
}

void write_vertices(FILE *fp, long radius,long nbmer, long nbpar)
{
  int i,j;
  float cur_radius;
  float phi,theta;
  Point3d vertex;

  // Calculate top point of sphere

  top.x = 0;
  top.y = radius;
  top.z = 0;
  write_vertex_to_file(fp,top);

  // Calculate vertices

  for (j=0;j<=nbmer;j++)
  {
    theta = j*2*PI/nbmer;
    for (i=0;i<nbpar;i++)
    {
      phi = (i+1)*PI/(nbpar+1);
      cur_radius = (float) radius * sin(phi);
      vertex.x = (long) (cur_radius * cos(theta));
      vertex.z = (long) (cur_radius * sin(theta));
      vertex.y = (long) (radius * cos(phi));    // y is up
      write_vertex_to_file(fp,vertex);
```

```
    }
  }

  // Create bottom point of sphere

  bottom.x = 0;
  bottom.y = -radius;
  bottom.z = 0;
  write_vertex_to_file(fp,bottom);
}

void write_polygons(FILE *fp, long nbmer, long nbpar, char *color)
{
  int i,j;
  int point;

  // Top of sphere

  for (j=0;j<nbmer;j++)
  {
    point = 1+nbpar*j;
    fprintf(fp,"%s 3 ",color);
    fprintf(fp,"0 %d %d\n",point,point+nbpar);
  }

  // Middle of sphere

  for (j=0;j<nbmer;j++)
  {
    for (i=0;i<nbpar-1;i++)
    {
      point = nbpar*j+i+1;
      fprintf(fp,"%s 4 ",color);  // 4 vertices per polygon
      fprintf(fp,"%d %d %d
 %d\n",point,point+1,point+nbpar+1,point+nbpar);
    }
  }

  // Bottom of sphere

  for (j=0;j<nbmer;j++)
  {
    point = nbpar + j*nbpar;
    fprintf(fp,"%s 3 ",color);
    fprintf(fp,"%d %d %d\n",point,nbvertices-1,point+nbpar);
  }
}

void main(int argc, char *argv[])
{

  FILE *plgfile;
  long radius,nbmer,nbpar;
  char *color;

  if (argc<6)
  {
```

```
    printf("Usage: sphere <filename> <radius> <nbmer> <nbpar>
  <color>\n");
    printf("writes a sphere to a plg file with given radius, number
  of\n");
    printf("meridians, number of parallels, with given color code.\n");
    exit(0);
  }

  if ((plgfile = fopen(argv[1],"w")) == NULL)
  {
    printf("Unable to open file for writing!\n");
    exit(1);
  }

  radius = atol(argv[2]);
  nbmer = atol(argv[3]);
  nbpar = atol(argv[4]);
  color = argv[5];

  printf("Creating Sphere with radius of %d\n",radius);
  printf("%d meridians, %d parallels, using %s color
 code",nbmer,nbpar,color);
  printf("\n");

  write_header(plgfile,nbmer,nbpar);
  write_vertices(plgfile,radius,nbmer,nbpar);
  write_polygons(plgfile,nbmer,nbpar,color);

  fclose(plgfile);

}
```

Exemple de fichier généré par `sphere.C`

```
Sphere 47 48
0 1000 0
500 866 0
866 499 0
1000 0 0
866 -500 0
500 -866 0
353 866 353
612 499 612
707 0 707
612 -500 612
353 -866 353
0 866 499
0 499 866
0 0 999
0 -500 866
0 -866 500
-353 866 353
-612 499 612
-707 0 707
-612 -500 612
-353 -866 353
-499 866 0
-866 499 0
-999 0 0
-866 -500 0
-500 -866 0
-353 866 -353
-612 499 -612
-707 0 -707
-612 -500 -612
-353 -866 -353
0 866 -499
0 499 -866
0 0 -999
0 -500 -866
0 -866 -500
353 866 -353
612 499 -612
707 0 -707
612 -500 -612
353 -866 -353
499 866 0
866 499 0
999 0 0
866 -500 0
500 -866 0
0 -1000 0
(suite)
```

```
0x1AFF 3 0 1 6
0x1AFF 3 0 6 11
0x1AFF 3 0 11 16
0x1AFF 3 0 16 21
0x1AFF 3 0 21 26
0x1AFF 3 0 26 31
0x1AFF 3 0 31 36
0x1AFF 3 0 36 41
0x1AFF 4 1 2 7 6
0x1AFF 4 2 3 8 7
0x1AFF 4 3 4 9 8
0x1AFF 4 4 5 10 9
0x1AFF 4 6 7 12 11
0x1AFF 4 7 8 13 12
0x1AFF 4 8 9 14 13
0x1AFF 4 9 10 15 14
0x1AFF 4 11 12 17 16
0x1AFF 4 12 13 18 17
0x1AFF 4 13 14 19 18
0x1AFF 4 14 15 20 19
0x1AFF 4 16 17 22 21
0x1AFF 4 17 18 23 22
0x1AFF 4 18 19 24 23
0x1AFF 4 19 20 25 24
0x1AFF 4 21 22 27 26
0x1AFF 4 22 23 28 27
0x1AFF 4 23 24 29 28
0x1AFF 4 24 25 30 29
0x1AFF 4 26 27 32 31
0x1AFF 4 27 28 33 32
0x1AFF 4 28 29 34 33
0x1AFF 4 29 30 35 34
0x1AFF 4 31 32 37 36
0x1AFF 4 32 33 38 37
0x1AFF 4 33 34 39 38
0x1AFF 4 34 35 40 39
0x1AFF 4 36 37 42 41
0x1AFF 4 37 38 43 42
0x1AFF 4 38 39 44 43
0x1AFF 4 39 40 45 44
0x1AFF 3 5 46 10
0x1AFF 3 10 46 15
0x1AFF 3 15 46 20
0x1AFF 3 20 46 25
0x1AFF 3 25 46 30
0x1AFF 3 30 46 35
0x1AFF 3 35 46 40
0x1AFF 3 40 46 45
```

Annexe C Bibliothèque de fonctions: objets

`void add_object_to_world(OBJECT *obj)`

Arguments: `obj`, objet à rajouter au monde virtuel
Valeur de retour: Aucune
Définition dans: `world.C`
Fichier include: `vr_api.h`

Description: Comme son nom l'indique, cette fonction rajoute un objet au monde virtuel. Même si un objet est chargé avec la fonction `load_plg_object`, il n'apparaîtra dans le monde que si les fonctions `update_object` et `add_object_to_world` sont appelées.

Voir aussi: `update_object`

`void attach_object(OBJECT *obj, OBJECT *to, BOOL preserve)`

Arguments: `obj`, objet auquel on attache l'autre objet
`to`, objet à rattacher
`preserve`, valeur booléenne
Valeur de retour: Aucune
Définition dans: `segment.C`
Fichier include: `vr_api.h`

Description: Cette fonction attache l'objet donné par `*obj` à un autre objet donné par `*to`. Cet objet `*to` devient parent de l'autre.

Voir aussi: `detach_object`

`void compute_object(OBJECT *)`

Arguments: `obj`, objet à évaluer
Valeur de retour: Aucune
Définition dans: `objsppt.C`
Fichier include: `vr_api.h`

Description: Les paramètres de l'objet sont mis à jour.
Voir aussi: `update_object`

`void detach_object(OBJECT *obj, BOOL preserve)`

Arguments: `obj`, objet à détacher
Valeur de retour: Aucune
Définition dans: `segment.C`
Fichier include: `vr_api.h`

Description: L'objet donné est détaché de son parent, s'il existe. Si l'objet n'a aucun parent, la fonction n'a aucun effet

Voir aussi: `attach_object`

`void do_for_all_objects(void (*fn)(OBJECT *));`

Arguments: `fn`, fonction à appeler
Valeur de retour: Aucune
Définition dans: `splits.C`
Fichier include: `vr_api.h`

Description: La fonction `fn` est appelée pour tous les objets contenus dans le monde virtuel.

`void do_for_all_selected(void (*fn)(OBJECT *));`

Arguments: `fn`, fonction à appeler
Valeur de retour: Aucune
Définition dans: `splits.C`
Fichier include: `vr_api.h`

Description: La fonction `fn` est appelée pour tous les objets sélectionnés.

`OBJECT *find_object_on_screen(WORD x, WORD y)`

Arguments: `x,y`, coordonnées à l'intérieur de la fenêtre
Valeur de retour: NULL si aucun objet n'a été trouvé ou pointeur à un objet
Définition dans: `cursor2d.C`
Fichier include: `vr_api.h`

Description: Cette fonction est utile pour trouver quel objet a été sélectionné avec la souris. Elle recherche un objet aux coordonnées `x,y` de la fenêtre de Mirage.

Voir aussi: `highlight_object, unhighlight_object`

`long get_object_bounds(OBJECT *obj, long *x, long *y, long *z)`

Arguments: `obj`, objet
`x,y,z`, adresses des variables coordonnées de l'objet
Valeurs de retour : rayon de la sphère englobant l'objet
`x,y,z`, centre de la sphère englobant l'objet
Définition dans: `objsppt.C`
Fichier include: `vr_api.h`

Description: Afin de faire des tests de collision entre objets, on peut utiliser la sphère englobante de l'objet. Cette fonction retourne le rayon et les coordonnées du centre de la sphère via les pointeurs `x,y` et `z`.

`void get_obj_info(OBJECT *obj, int *nv, int *np);`

Arguments: obj, objet
nv, pointeur au nombre de sommets
np, pointeur au nombre de polygones
Valeurs de retour : nv, np: nombre de sommets et polygones de l'objet
Définition dans: objsppt.C
Fichier include: vr_api.h

Description: Cette fonction récupère le nombre de sommets et de polygones contenus dans l'objet donné en argument.

void get_object_pose(OBJECT* obj, POSE *p)

Arguments: obj, objet
p, adresse d'une variable de type POSE
Valeur de retour: p, pose de l'objet donné en argument dans son propre référentiel
Définition dans: segment.C
Fichier include: vr_api.h

Description: Cette fonction est utile pour obtenir la position et les angles de rotation d'un objet dans le référentiel de l'objet.

Voir aussi: set_object_pose, get_object_world_pose

void get_object_world_pose(OBJECT *obj, POSE *p)

Arguments: obj, objet
p, adresse d'une variable de type POSE
Valeur de retour: p, pose de l'objet donné en argument dans le référentiel global
Définition dans: segment.C
Fichier include: vr_api.h

Description: Cette fonction place dans une variable de type POSE la position et les angles de rotation d'un objet dans le référentiel global.

Voir aussi: set_object_world_pose, get_object_world_position

void get_object_world_position(OBJECT *obj, long *x, long *y, long *z)

Arguments: obj, objet
x, y, z, adresses des variables de retour
Valeurs de retour: x, y, z, coordonnées de l'objet dans le référentiel global
Définition dans: segment.C
Fichier include: vr_api.h

Description: Cette fonction place dans les variables x, y et z la position de l'objet donné dans le référentiel global.

Voir aussi: get_object_world_pose

void highlight_object(OBJECT *obj)

Arguments: obj, objet à mettre en évidence
Valeur de retour: Aucune
Définition dans: objsppt.C
Fichier include: vr_api.h

Description: L'objet donné en argument est mis en évidence en traçant les bordures de ses polygones en blanc pour montrer qu'il est sélectionné.

Voir aussi: unhighlight_object

BOOL is_object_selectable(OBJECT *obj)

Arguments: obj, objet
Valeur de retour: 1 si l'objet peut être sélectionné, sinon 0
Définition dans: segment.C
Fichier include: vr_api.h

Description: La fonction retourne une valeur 1 si l'objet peut être sélectionné. Un objet représentant un curseur 3D est un exemple d'objet qui ne devrait pas être sélectionné.

Voir aussi: make_object_selectable

BOOL is_object_selected(OBJECT *obj)

Arguments: obj, objet
Valeur de retour: 1 si l'objet est sélectionné, sinon 0
Définition dans: segment.C
Fichier include: vr_api.h

Description: La fonction retourne 1 si l'objet a été sélectionné, sinon 0.

Voir aussi: highlight_object, unhighlight_object

OBJECT *load_plg_object(FILE *in, POSE *p, float sx, float sy, float sz, WORD depth)

Arguments: in, pointeur d'un fichier ouvert de type plg
p, pose à donner à l'objet
sx, sy, sz, facteur d'échelle à donner à l'objet
depth, façon dont l'objet sera traité lors du processus de rendu réaliste
Valeur de retour: Pointeur à l'objet chargé, NULL si le chargement a échoué
Définition dans: objfile.C
Fichier include: vr_api.h

Description: Cette fonction charge un objet de type PLG à partir d'un fichier. Le fichier doit être préalablement ouvert avec la fonction fopen() en mode lecture. La position et l'orientation sont données par le paramètre p. Les valeurs sx,sy et sz sont les facteurs d'échelle.

Voir aussi: update_object, add_object_to_world

OBJECT *load_nff_object(FILE *in, POSE *p, float sx, float sy, float sz, WORD depth)

Arguments: in, pointeur d'un fichier ouvert de type nff
p, pose à donner à l'objet
sx,sy,sz, facteur d'échelle à donner à l'objet
depth, facon dont l'objet sera traité lors du processus de rendu réaliste
Valeur de retour: Pointeur à l'objet chargé, NULL si le chargement a échoué
Définition dans: objfile.C
Fichier include: vr_api.h

Description: La fonction charge un objet de type NFF. Voir la fonction précédente pour plus d'informations.

Voir aussi: load_plg_object

void remove_object_from_world(OBJECT *obj)

Arguments: obj, objet à enlever du monde virtuel
Valeur de retour: Aucune
Définition dans: world.C
Fichier include: vr_api.h

Description: Cette fonction retire un objet du monde virtuel. L'objet n'est pas effacé de la mémoire. Il peut donc être rappelé ultérieurement avec add_object_to_world.

Voir aussi: add_object_to_world

void save_plg(OBJECT *obj, FILE *out, BOOL world)

Arguments: obj, objet à sauvegarder
out, pointeur du fichier à sauvegarder
world, 1 pour sauver en coordonnées globales, 0 pour référentiel de l'objet
Valeur de retour: Aucune
Définition dans: world.C
Fichier include: vr_api.h

Description: Cette fonction sauvegarde un objet dans un fichier PLG. Le fichier doit être préalablement ouvert avec la fonction fopen() en mode écriture. Les coordonnées seront soit dans le référentiel de l'objet, soit dans le référentiel global selon l'argument world.

Voir aussi: do_for_all_selected

void select_next_representation(OBJECT *obj)

Arguments: obj, objet
Valeurs de retour : Aucune
Définition dans: objsppt.C
Fichier include: vr_api.h

Description: La représentation suivante de l'objet est activée. Si la représentation active est la dernière, la première représentation est choisie.

Voir aussi: `select_first_representation`

`void set_object_pose(OBJECT *obj, POSE *p)`

Arguments: `obj`, objet à modifier
`p`, pose à donner à l'objet dans son propre référentiel
Valeur de retour: Aucune
Définition dans: `segment.C`
Fichier include: `vr_api.h`

Description: Cette fonction donne une position et orientation à un objet. Pour que les changements soit visibles, on doit appeler la fonction `update_object`.

Voir aussi: `set_object_world_pose, get_object_pose`

`void set_object_world_pose(OBJECT *obj, POSE *p)`

Arguments: `obj`, objet à modifier
`p`, `POSE` à donner à l'objet dans le référentiel global
Valeur de retour: Aucune
Définition dans: `segment.C`
Fichier include: `vr_api.h`

Description: Identique à la fonction précédente, sauf que le réferentiel global est utilisé.

Voir aussi: `set_object_pose`

`void unhighlight_object(OBJECT *obj)`

Arguments: `obj`, objet à désélectionner
Valeur de retour: Aucune
Définition dans: `objsppt.C`
Fichier include: `vr_api.h`

Description: Cette fonction désélectionne un objet. Les bordures des polygones de l'objet ne seront plus tracées en blanc.

Voir aussi: `highlight_object`

`void update_object(OBJECT *obj)`

Arguments: `obj`, objet mis à jour
Valeur de retour: Aucune
Définition dans: `segment.C`
Fichier include: `vr_api.h`

Description: Cette fonction fait la mise à jour de l'objet. Elle est généralement appelée après une fonction du type `set_object_pose`.

Annexe D Bibliothèque de fonctions: caméras et téléportations

`compute_camera_factors(CAMERA *c)`

Arguments: `c`, caméra à mettre à jour
Valeur de retour: Aucune
Définition dans: `scamera.C`
Fichier include: `vr_api.h`

Description: Cette fonction fait la mise à jour des paramètres associés aux caméras. Elle doit être appelée après `set_camera_window` par exemple.

`TELEPORT *create_teleport()`

Arguments: Aucun
Valeur de retour: Pointeur au point de téléportation créé
Définition dans: `world.C`
Fichier include: `vr_api.h`

Description: La fonction alloue un espace mémoire pour une variable de type `TELEPORT` et l'initialise.

`void delete_teleport(TELEPORT *)`

Arguments: `t`, point de téléportation à effacer
Valeur de retour: Aucune
Définition dans: `world.C`
Fichier include: `vr_api.h`

Description: Cette fonction libère la mémoire occupée par une variable de type TELEPORT. Elle est utile uniquement si la variable a été allouée de façon dynamique.

`void get_camera_worldpose(CAMERA *c, POSE *p)`

Arguments: `c`, caméra
`p`, adresse de la variable de retour
Valeur de retour: `p`, Pose de la caméra donnée
Définition dans: `scamera.C`
Fichier include: `vr_api.h`

Description: Cette fonction récupère la position et les angles de rotation d'une caméra dans le référentiel global.

`SCALE get_camera_zoom(CAMERA *c)`

Arguments: `c`, caméra
Valeur de retour: Effet de loupe [*zoom*] de la caméra donnée
Définition dans: `scamera.C`
Fichier include: `vr_api.h`

Description: Cette fonction récupère la valeur de l'effet de loupe de la caméra donnée.

Voir aussi: `set_camera_zoom`

`void set_camera_hither(CAMERA *c, COORD h)`

Arguments: `c`, caméra
`h`, valeur de la distance du plan avant du parallélépipède de vision
Valeur de retour: Aucune
Définition dans: `scamera.C`
Fichier include: `vr_api.h`

Description: Cette fonction règle la distance de découpage du plan avant du parallélépipède de vision. Les objets à une distance inférieure ne seront pas affichés. Le découpage, qui fait partie du processus de rendu réaliste, est expliqué au chapitre 3.

Voir aussi: `set_camera_yon`

`void set_camera_window(CAMERA *c, WORD l, WORD t, WORD r, WORD b)`

Arguments: `c`, caméra
`l,t`, coordonnées x,y du coin supérieur gauche de la fenêtre de visualisation
`r,b` coordonnées x,y du coin inférieur droit de la fenêtre de visualisation
Valeur de retour: Aucune
Définition dans: `scamera.C`
Fichier include: `vr_api.h`

Description: Cette fonction crée une fenêtre de visualisation à l'intérieur de la clôture de Mirage (*viewport*).

`void set_camera_yon(CAMERA *c, COORD y)`

Arguments: `c`, caméra
`y`, valeur de la distance du plan arrière du parallélépipède de vision
Valeur de retour: Aucune
Définition dans: `scamera.C`
Fichier include: `vr_api.h`

Description: Cette fonction ajuste la distance du plan arrière [*yon*]. Les objets plus distants ne seront pas affichés. Le découpage, qui fait partie du processus de rendu réaliste, est expliqué au chapitre 3.

Voir aussi: `set_camera_hither`

`void set_camera_zoom(CAMERA *c, SCALE zoom)`

Arguments: `c`, caméra
`zoom`, valeur de l'effet de loupe à donner à la caméra
Valeur de retour: Aucune
Définition dans: `scamera.C`
Fichier include: `vr_api.h`

Description: Cette fonction règle l'effet de loupe de la caméra donnée.

Voir aussi: `get_camera_zoom`

`void teleport_set_here(TELEPORT *t, POSE *p)`

Arguments: `t`, point de téléportation
`p`, position et orientation de la caméra à donner au point de téléportation
Valeur de retour: Aucune
Définition dans: `world.C`
Fichier include: `vr_api.h`

Description: Cette fonction sauvegarde un point de téléportation. Pour sauvegarder la position actuelle, on donne la variable `current_pose` en argument.

Voir aussi: `teleport_to`

`void teleport_set_vehicle(TELEPORT *t, OBJECT *vehicle, char *vname)`

Arguments: `t`, point de téléportation
`vehicle`, véhicule ou objet auquel se rattache le point de téléportation
`vname`, nom du véhicule
Valeur de retour: Aucune
Définition dans: `world.C`
Fichier include: `vr_api.h`

Description: Le point de téléportation est attaché à un objet véhicule. Si l'objet se déplace, le point de téléportation suivra.

Voir aussi: `create_teleport`

`void teleport_to(TELEPORT *t)`

Arguments: `t`, point de téléportation
Valeur de retour: Aucune
Définition dans: `world.C`
Fichier include: `vr_api.h`

Description: Cette fonction téléporte l'usager au point de téléportation donné.

Voir aussi: `create_teleport`, `teleport_set_here`

Annexe E Animation d'un cube tournant réalisé par fichier de type WLD

```
# cube.wld - Animation d'un cube tournant
# Christophe Maurel, 8 mars 1997

hither            10     # Distance de "clipping" proche
yon          1000000     # Distance de "clipping" lointaine

start 0,0,-500 0,0,0 1  # Position de depart, orientation et zoom

loadpath plg    # les objets se trouvent dans ce repertoire

#Positionner le cube
object cube=block.plg   1,1,1  0,0,0   0,50,0     0

animation 15    # maximum 15 pas/seconde

  state tourne
    do cube=step(0,0,0 4,10,5)[][]
```

Annexe F La compilation, l'optimisation et le déboguage

Le fichier `makefile` de Mirage

Voici le `makefile` de Mirage:

```
.SUFFIXES: .C .o
INCLUDE_DIRS = -I/usr/local/include -I/usr/include/X11 -
 Iinclude\
               -I/usr/local/globe/include/local
DEBUG    =                # -g: genere du code de debogage
PROFILE  =                # -p: genere du code pour profils
OPTIMIZE = -O3                 # -O: optimise le code
CC       = gcc                 # compilateur C
C++      = g++                 # compilateur C++ utilise
TKLIBS   = -ltk4.2 -ltcl7.6    # librairies Tcl/Tk
XLIBS    = -lXt -lX11 -lm      # librairies X windows
.C.o:
        $(C++) -c $(DEBUG) $(PROFILE) $(OPTIMIZE)
 $(INCLUDE_DIRS) $<

Obj =   Xgraphics.o Xevents.o Xmouse.o Xkeyboard.o ...


Src =   Xgraphics.C Xevents.C Xmouse.C Xkeyboard.C ...

mirage:  $(Obj)
        $(C++) -o $@   $(Obj)  $(TKLIBS) $(XLIBS)
        @echo " ------- Fini -------"
depend :
        makedepend -- $(INCLUDE_DIRS) -- $(Src)
clean:
        rm -f core *.o *~ mirage
# DO NOT DELETE THIS LINE -- make depend depends on it.
```

Figure F.1 `makefile` de Mirage (abrégé)

Optimisations

Afin d'optimiser l'exécution d'un programme UNIX, il est recommandé d'abord d'utiliser l'option -O3 du compilateur g++ ou CC (OPTIMIZE = -O3 dans le makefile), qui produira alors un fichier exécutable optimisé. Ainsi, Mirage compilé avec l'option -O3 est environ 30% plus performant sur Linux.

Il est également possible d'obtenir un profil de l'exécution du programme avec l'option -p du compilateur (PROFILE = -p dans le makefile). Lorsque le programme est exécuté, un fichier descriptif mon.out est généré, donnant l'état du programme toutes les 0.01 secondes. Un autre programme tel que gprof sur Linux (ou prof sur SGI) utilise ces informations pour présenter un profil d'exécution. Donc, si Mirage est compilé avec l'option -p, puis exécuté, on peut obtenir le profil avec : gprof mirage. La sortie est alors de la forme suivante:

```
Flat profile:
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 28.71     17.31    17.31                    z_output(nV *)
  9.72     23.17     5.86                    xy_transform(VERTEX *)
  7.60     27.75     4.58                    z_convert_vertex(VERTEX
 *)
  5.85     31.28     3.53                    find_normal(long, long,
 long, long, long, long, long, long, long, long *, long *, long *)
  5.11     34.36     3.08                    light_cosine(long, long,
 long, long, long, long, long, long, long)
  4.64     37.16     2.80                    is_poly_facing(_poly *)
  4.53     39.89     2.73                    proc_poly(_poly *)
  3.88     42.23     2.34                   unpack_poly_vertices(NPOLY
 *, int *, int)
  3.25     44.19     1.96                    compute_normal(_poly *)
  3.25     46.15     1.96                    zclip_loop(_poly *)
  2.57     47.70     1.55                    fastpoly(int, int *, int)
  2.50     49.21     1.51                   polsort(void const *,
 void const *)
  2.24     50.56     1.35                    magnitude32(long, long,
 long)
```

Figure F.2 Sortie de la commande `gprof mirage` (abrégé)

Les fonctions occupant le plus de ressources du processeur sont celles qu'il faudra tenter d'optimiser. La fonction `z_output()` par exemple est à considérer. Il est parfois possible de remplacer des fonctions ou opérations par un équivalent plus performant. Par exemple, les fonctions `sin()` et `sqrt()` peuvent être remplacées par des recherches dans un tableau ayant de nombreuses valeurs pré-calculées (*look-up table*). Le résultat ne sera pas exact mais il sera obtenu beaucoup plus rapidement. Étant donné que la performance de Mirage influe directement sur la perception d'immersion dans un monde virtuel, l'étape d'optimisation dans le développement de Mirage (ou tout autre logiciel de RV) est tout à fait fondamentale.

Le déboguage de Mirage

La recherche d'erreurs dans le code est facilitée par l'outil de « débogage » `gdb`, existant sur toutes les plates-formes UNIX. Mirage doit être d'abord compilé avec l'option `-g` (`DEBUG = -g` dans le `makefile`). L'exécutable comprend alors des références dont `gdb` se sert. Cet outil peut être utilisé en combinaison avec l'éditeur emacs. Gdb est appelé sous Emacs avec la commande « `Esc-x gdb` » (en notation Emacs `M-x gdb`). En ouvrant une deuxième fenêtre dans Emacs avec la commande « `Ctrl-x 2` », on peut alors exécuter le code pas à pas avec la commande « `next` » de `gdb` en visualisant l'exécution dans le code source dans la deuxième fenêtre. Le programme s'arrête à tous

les points d'arrêt [*breakpoints*] identifiés par la commande « `break` » (ex: `break create_teleport`).

Sous Linux, une version de `gdb` possédant une interface graphique peut être utilisée: `xxgdb` (voir Figure F.3). Cet outil permet d'ouvrir une fenêtre séparée pour visualiser l'exécution dans le code source tel que vue dans la Figure F.4.

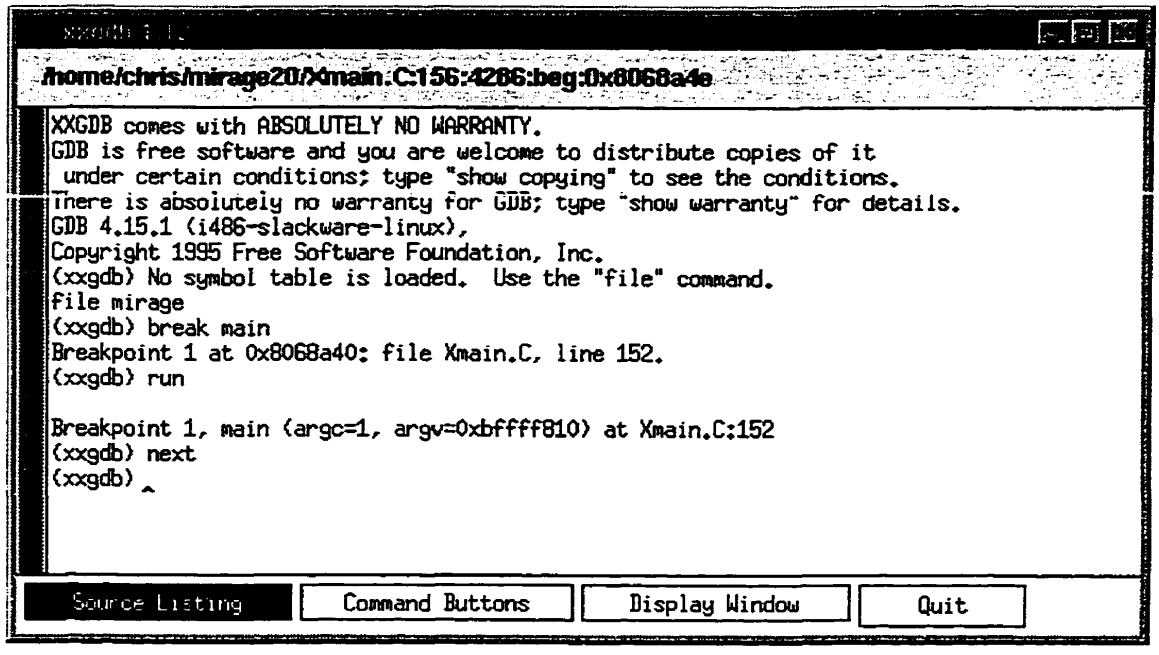


Figure F.3 Fenêtre principale de xxgdb

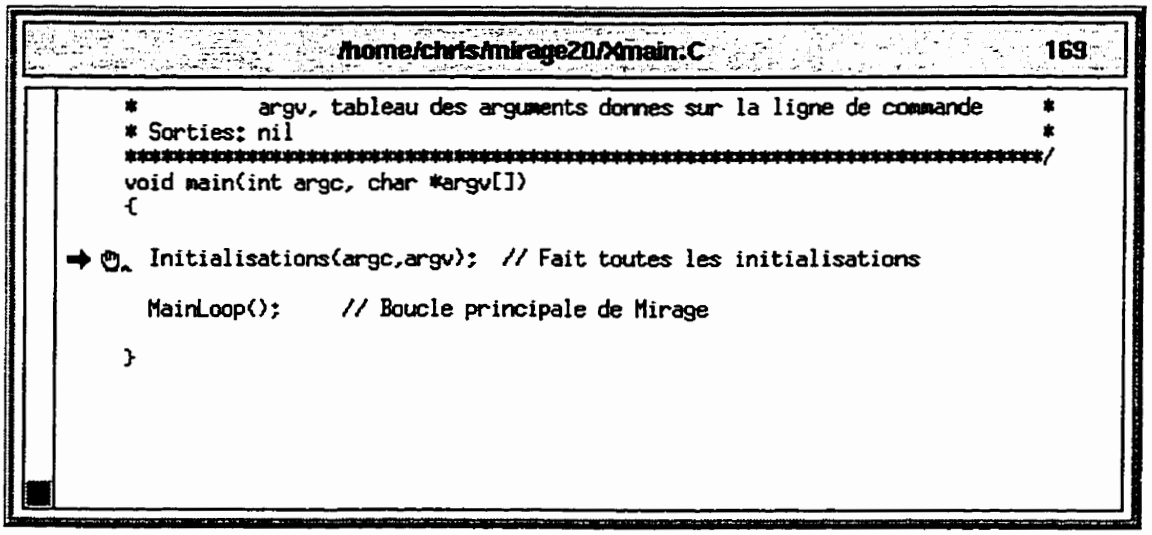


Figure F.4 Vue du code source de Mirage dans xxgdb

Annexe G Code source de `Xmain.C`

```
// ----------------------------------------------
// Xmain.C
//
// Auteur: Christophe Maurel
//
// Mirage 2.0
// Christophe Maurel
// 23 Janvier 1997
//
// ----------------------------------------------

/* This code is part of Mirage, created by Christophe Maurel from VR386 by
   Dave Stampe. VR386 is descended from REND386, by Dave Stampe and Bernie
   Roehl.
   This program can be used for educational purposes or to create a public
   domain application. Any person wishing to develop a commercial application
   must ABSOLUTELY contact Christophe Maurel (maurel@electech.polymtl.ca).
   If you use this code for your application, you must mention Mirage and
   its author Christophe Maurel.

   Ce code fait partie de Mirage, cree par Christophe Maurel a partir de VR386
   de Dave Stampe. VR386 est un descendant de REND386, cree par Dave Stampe et
   Bernie Roehl.
   Ce programme peut etre utilise a des fins pedagogiques ou pour une
   application du domaine public. Toute personne voulant developper une
   application commerciale a partir de ce code doit ABSOLUMENT contacter
   Christophe Maurel (maurel@electech.polymtl.ca).
   Si vous utilisez ce code pour votre application, vous devez mentionner
   Mirage et l'auteur Christophe Maurel.

   If you have any questions or comments, please contact:
   Si vous avez des questions ou commentaires, vous pouvez contacter :

   Christophe Maurel, maurel@electech.polymtl.ca
*/

#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include <stdio.h>
#include <stdlib.h>
#include <tcl.h>
#include <tk.h>

#include "vr_api.h"
```

```
#include "oldtasks.h"
#include "misc.h"

/************** Definitions ***************/

#define XEVENTS (KeyPressMask | ButtonPressMask | ButtonReleaseMask | ExposureMask |
StructureNotifyMask | EnterWindowMask)

typedef void (*TreatEv); // type des fonctions "handler" d'evenements

/************** Variables globales ********/

XEvent event;

BOOL running = 0;
BOOL in_graphics = 0;

TASK *tasklist = NULL;

int mouse_nav = 1;          // use mouse as joystick
int mouse_motion = 0;  // mouse button not down
int page_intro = 0;
int flymode = 0;

int animatemode = 1;    /* if set, animation is on */
int do_horizon = 1;     /* if set, draw a horizon  */
int show_location = 1;  /* if set, we display current location on-screen */
int show_compass = 1;   /* if set, we display 3-D compass on-screen */
int show_framerate = 1; /* if set, we display frames/second rate */
int do_screen_clear = 1; /* by default, we clear the screen on each frame */
int use_frame = 0;      /* if set, draw a "frame" */
int wireframe = 0;      /* if set, draw objects in wireframe */


/********* Fonctions externes ***********/

void DoTkCommand();
void PlaybackDemoStep();
void RecordHandlers();
void TreatEvent(XEvent event);
void mirage_init();
void mirage_update();

/********* Variables externes ***********/

extern Display *display;
extern Colormap colormap;


/*****************************************************************************
* refresh_display:  Fait la mise a jour de fenetre de Mirage lorsque le    *
```

```
*              point de vue a change, un objet a change de position,  *
*              etc..                                                  *
*-------------------------------------------------------------------*
* Entrees: nil                                                       *
* Sorties: nil                                                       *
*********************************************************************/
void refresh_display()
{
 update_body_links();
 screen_refresh(current_camera);

 position_changed = 0;
 world_changed = 0;
 display_changed = 0;
}

/*********************************************************************
* Initialisations:  Appelle les diverses routines d'initialisation de *
*              Mirage, ouvrant les fichiers donnes en parametre,     *
*              ouvrant la fenetre X, enregistrant les "handlers", ... *
*-------------------------------------------------------------------*
* Entrees: argc, nombre d'arguments de la ligne de commande          *
*        argv, tableau des arguments donnes sur la ligne de commande *
* Sorties: nil                                                       *
*********************************************************************/
void Initialisations(int argc,char *argv[])
{
 title_screen();
 preload_initialize(argc, argv);
 create_default_segs();              // for animations
 create_default_lights();           // for world loading
 read_input_files(argc, argv);      // get any files to be loaded
 atexit(exit_handler);
 video_initialize();
 device_initialize();
 RecordHandlers();                  // records event handlers
 mirage_init();                    // initialisations a rajouter
}

/*********************************************************************
* MainLoop: Boucle principale de Mirage, qui verifie la presence     *
*        d'evenements X Windows et appelle la fonction "handler"     *
*        appropriee. Les routines d'animation sont aussi appellees,  *
*        et les evenements de la fenetre Tcl/Tk sont traites.        *
*-------------------------------------------------------------------*
* Entrees: nil                                                       *
* Sorties: nil                                                       *
*********************************************************************/
void MainLoop()
{
 int flags = TK_DONT_WAIT;
```

```
running = 1;

while (running)
{
 PlaybackDemoStep();

 if (!XCheckMaskEvent(display, XEVENTS , &event))
 {
  if (mouse_motion && mouse_nav)
        if (process_mouse_click(event,1))
         position_changed++;;
 }
 else
  TreatEvent(event);

 // animation
 if (animatemode==-1) animatemode = 0; /* single step OFF */
 if (animatemode && page_intro)
 {
  run_tasks(tasklist);          // tasks programmes dans WLD
  do_animations();              // animations programmees dans WLD
  mirage_update();              // animations programmees en C
 }

 // update screen if needed
 if (position_changed || display_changed || world_changed)
  refresh_display();

 XFlush(display);
 Tk_DoOneEvent(flags);
 if (page_intro != 0) // have we read the script yet ?
  DoTkCommand();  // execute menu command if available
 }
}

/*****************************************************************************
* main: Fonction principale de Mirage, appelant la fonction d'initiali-    *
*      sation et la boucle principale de Mirage                             *
*---------------------------------------------------------------------------*
* Entrees: argc, nombre d'arguments de la ligne de commande                 *
*         argv, tableau des arguments donnes sur la ligne de commande       *
* Sorties: nil                                                              *
*****************************************************************************/

void main(int argc, char *argv[])
{

 Initialisations(argc,argv);  // Fait toutes les initialisations
 MainLoop();       // Boucle principale de Mirage

}
```

Annexe H Les fichiers de Mirage

Fichiers X Windows

Tableau H.1 Fichiers X Windows

Fichier	Utilité
XXmouse.C	Contient les routines pour la manipulation avec la main virtuelle
Xevents.C	Gestion des événements par des fonctions *handler*
Xgraphics.C	Fonctions graphiques : ouverture de fenêtre, polygone, texte, ...
Xkeyboard.C	Lecture du clavier
Xmain.C	Initialisations et boucle principale de Mirage
Xmouse.C	Gestion des événements venant de la souris

Fichiers ayant attrait au rendu réaliste

body.C,colormap.C,cursor2d.C,dos.C,emmfsppt.C,filesppt.C,
gloveptr.C,grabbing.C,horizon.C,hormath.C,joyptrs.C,
keyboard.C,lighting.C,lights.C,logiptr.C,manip3d.C,
mathinit.C,matrix.C,memory.C,mirage.C,miscmath.C,
mouseptr.C,navjoy.C,objrend.C,objsppt.C,pointer.C,
refresh.C,rendcore.C,scamera.C,segment.C,sgsppt.C,splits.C,
sun.C,texture.C,timer.C,title.C,uscreen.C,userint.C,
uservid.C,viewpcx.C,viewrend.C

Fichiers gestionnaires de fichiers de type PLG/FIG/WLD, DXF et NFF

demotask.C,dxf2plg.C,init.C,loadfig.C,objfile.C,readcfg.C,
statmach.C,tasks.C,world.C,wparse.C

Fichiers Tcl/Tk

Ces fichiers implantent l'interface Tcl/Tk de Mirage:

Tableau H.2 Fichiers de l'interface Tcl/Tk

Fichier	Utilité
tkcommands.C	Fonctions appelées par le menu Tk
tkdemo.C	Fonctions reliées au menu Démo
tkmenu.C	Gestion des menus Tk
tksupport.C	Fonctions de support à tkcommands.C

Fichiers assembleurs de VR386 et équivalences dans Mirage

Les fichiers assembleurs de VR386 effectuant le processus de rendu réaliste ont du être portés à Mirage. Ces fichiers ont été recréés en langage C. La liste est donnée ci-dessous.

Tableau H.3 Les fichiers assembleurs de VR386 avec leurs équivalences en C pour Mirage

Fichiers assembleur	Fichiers C correspondants	Fonction du fichier
animate.asm	animate.C	Animation des objets
intrig.asm	intrig.C	Fonctions de trigonométrie
lighting.asm	lighting.C	Calcul des lumières
objrend.asm	objrend.C	Rendu des objets
pwrglvio.asm	-	Interface Powerglove (DOS)
viewrend.asm	viewrend.C	Rendu 3D
hormath.asm	hormath.C	Fonctions pour l'horizon
intsplit.asm	intsplit.C	Fonctions pour les *splits*
matrixm.asm	matrixm.C	Fonctions matricielles
polyout.asm	polyout.C	Rendu des polygones
sqrti.asm	sqrti.C	Fonctions racines carrées, norme de vecteur
xyclip.asm	xyclip.C	Découpage x et y sur la fenêtre
int3d.asm	int3d.C	Fonctions de manipulation d'objets
joytimer.asm	-	Interface joystick (DOS)
miscmath.asm	miscmath.C	Fonctions mathématiques variées
polyproc.asm	polyproc.C	Rendu des polygones
vdrinte.asm	-	Pilote vidéo (DOS)

Fichiers de VR386 non portés à Mirage

Plusieurs fichiers de VR386 n'ont pas été portés à Mirage, la raison principale étant que ces fichiers contrôlent des périphériques connectés au port série sous DOS. Mirage ne supporte pas ces périphériques. Ces gestionnaires de périphériques pourraient servir à l'avenir pour faire leurs interfaces avec Mirage. Comme ces fichiers ne sont pas compilés avec Mirage, ils se trouvent dans le répertoire `vr386`. La liste est donnée dans la Tableau H.4.

Tableau H.4 Fichiers de VR386 non-convertis pour Mirage

Fichier	Ancienne utilité
`cursglov.C`	Gestion du Mattel Powerglove
`cursor3d.C`	Gestion des curseurs 3D
`drvload.C`	Chargement de divers drivers
`headtrak.C`	Gestion de *headtracker*
`hpotptr.C`	Gestion du joystick
`pcxfile.C`	Chargement d'images de type PCX

Annexe I Étapes dans le développement de Mirage

- Conversion du code de REND386 à un code entièrement en C.
- Écriture des fonctions X Windows pour REND386.
- Un prototype de REND386 sur UNIX est obtenu, mais ayant de nombreux problèmes.
- En raison de sérieux problèmes avec REND386 (code confus causant de grandes difficultés dans la conversion à UNIX), REND386 est abandonné pour VR386, une version améliorée de REND386, ayant un code mieux structuré mais plus long à porter en raison des fonctionnalités rajoutées.
- Conversion du code assembleur VR386 au langage C (utilisant TURBO C++ pour DOS). Le code est traduit et « débogué » module par module pendant trois mois.
- Une version de VR386 « déboguée » entièrement en langage C est obtenue.
- Les fonctions X Windows développées pour REND386 sont utilisées pour porter VR386 à UNIX. De nombreuses erreurs de compilation n'apparaissant pas sous DOS doivent être corrigées pour obtenir une version fonctionnant sur UNIX.
- Le nouveau logiciel est baptisé Mirage.

- Antoine D'Anjou rajoute des fonctions pour importer des fichiers de WorldToolKit dans Mirage, particulièrement une station d'ESOPE-RV en format `NFF`.

- Christian Auger rajoute des fonctions pour manipuler des objets avec une main virtuelle avec anti-collision entre objets.

- La capacité d'importer des objets de type `DXF` d'AutoCAD est rajoutée à Mirage.

- Une nouvelle interface utilisant Tcl/Tk est développée pour Mirage.

- La capacité d'enregistrer et rejouer les mouvements de l'usager est rajoutée à Mirage.

- Martin Clouâtre et Ravdeep Ahuja développent le support de manettes de jeux sur Linux pour se déplacer ou manipuler des objets virtuels.

- Le programme principal `Xmain.C` est restructuré en créant un fichier séparé `Xevents.C` avec des fonctions de rappel qui traitent les événements X.

Annexe J Commandes principales des fichiers WLD

```
addrep nomobj = nomfichier sx,sy,sz rx,ry,rz tx,ty,tz size
map
```

Additionne une nouvelle représentation à un objet

```
ambient n
```

Détermine la valeur de la lumière ambiante (0-127)

```
anglestep f
```

Spécifie la valeur du pas de rotation

```
animation f
```

Débute une animation, f: nombre de pas/seconde

```
area x,y,z nom
```

Déclare une zone délimitée par des *splits*

```
attach objet parent
```

Attache un objet à un objet parent

```
attachview parent
```

Attache le point de vue à un objet

```
camera n x,y,z tilt,pan,roll zoom hither,yon
```

Déclare une caméra, n: numéro de la caméra

```
depthtype objet n
```

Change la façon de dessiner l'objet

```
detach objet
```

Détache un objet de son objet parent

```
do nom=fonction [argument] [operation]
```

Effectue une opération dans une animation

`figure nom=nomfichier sz,sy,sz rx,ry,rz tx,ty,tz parent`

Charge une figure avec les paramètres donnés

`groundcolor n`

Détermine la couleur du sol

`hither n`

Détermine la distance de *clipping* proche

`if (test1 test2) nom=fonction [arguments] [operations]`

Effectue une opération d'une animation si le test est vérifié

`include nomfichier`

Inclue un fichier comme s'il était avait été rentré

`light x y z spot`

Déclare une source de lumière

`loadpath repertoire`

Spécifie un répertoire pour les fichiers

`mlight x y z spot intensité nom=parent`

Déclare une source de lumiere amovible

`object nom=nomfichier sx,sy,sz rx,ry,rz tx,ty,tz d map parent`

Charge un objet PLG avec les paramètres donnés

`polyobj nv couleur v1x,v1y,v1z, v2x,v2y,v2z, ...`

Crée un polygone, nv: nombre de vertex

`polyobj2 nv couleur1,couleur2 v1x,v1y,v1z, v2x, ...`

Crée deux polygones, face à face

`position objet x,y,z`

Positionne un objet aux coordonnées données

`surfacedef nomsurface valeur`

Définit un type de surface avec la valeur donnée

`surfacemap nom`

Définit une nouvelle *surfacemap*

`task nom période parametres`

Crée un *task* avec la période donnée

`window x,y,largeur,hauteur`

Spécifie une fenêtre de vue avec un coin et la hauteur et largeur

`worldscale f`

Spécifie l'échelle du monde virtuel, par défaut 1 unité = 1 millimètre

`yon n`

Spécifie la distance de *clipping* lointaine

Annexe K Considérations pour des mondes WLD de VR386

Bien que Mirage ait été développé avec l'intention de demeurer compatible avec VR386, les différences entre les systèmes d'exploitation DOS et UNIX peuvent causer certains problèmes. En effet, les fichiers textes de DOS ont un retour-de-chariot à la fin de chaque ligne du fichier tandis que les fichiers textes de UNIX n'en ont pas. Il est donc nécessaire de convertir les fichiers de types `PLG`, `FIG` et `WLD` au format UNIX avec un utilitaire du type `dos2unix` ou `to_unix`. Comme UNIX différencie la casse des caractères dans les noms des fichiers, il faut s'assurer que la casse des noms des fichiers mentionnés dans les fichiers de types `FIG` ou `WLD` correspond bien à celle des noms de fichiers réels. De plus, UNIX utilise la barre oblique « / » [*slash*] pour séparer les noms de répertoires à la différence du DOS qui utilise la barre oblique inversée « \ » [*backslash*]. Il faut donc effectuer la conversion de ces caractères pour éviter les erreurs de lecture.

A Low-Cost PC-Oriented Virtual Environment for Operator Training

E. K. Tam† C. Maurel‡ P. Desbiens‡ R. J. Marceau‡ A. S. Malowany† L. Granger‡

† Department of Electrical Engineering
McGill University
3480 University Street, Montreal
Quebec, Canada, H3A 2A7
{elaine, malowany}@ee.mcgill.ca

‡ Département de génie électrique et informatique
École Polytechnique de Montréal
2900, chemin Édouard-Montpetit, Montréal
Québec, Canada, H3C 3A7
{maurel, desbiens, marceau, granger}@electech.polymtl.ca

Abstract— **Virtual environment (VE) technologies have been shown to provide invaluable training in the performance of complex procedural tasks. Of paramount importance to the success of any VE is realism; entities must move and behave believably and approximate the complexity of the real world. Such real-world fidelity is accompanied by the challenge of making immersive training in virtual reality (VR) more widely-available, at costs significantly less prohibitive. A solution is presented to recreate the VE on low-cost PCs, reducing the cost of the VR software itself, yet preserving real-time performance. This paper presents a low-cost, PC-based VR development system within the framework of ESOPE-VR, a VR operator training simulator (OTS) prototype for power-utility personnel. PC-ESOPE-VR aims to provide the same or improved functionality and performance as the original, with a 3-D visual interface, voice recognition and feedback, navigation and manipulation facilities, and expert system, multimedia and multi-user (distributed) support, yet address the practical demands raised above. The PC-based system is also compatible with the original; it can draw on existing ESOPE-VR power component libraries and support the automatic synthesis of a virtual power station environment from its single-line diagram representation. The approach used towards fulfilling all these issues is described, followed by key implementation results.**

I. Introduction

The use of virtual environment (VE) technology as a training tool has shown promise in various domains due to its ability to offer intuitive modes of interaction analogous to the ways in which humans communicate with each other or manipulate objects in the real world [1]. Equally important is the ability of VEs to offer accessible and cost-effective means for training personnel who currently receive little or no experiential preparation for their assigned tasks [2]. Yet, the widespread use of VE technology for training is limited by virtual reality (VR) computing requirements, primarily in relation to hardware cost which, if not prohibitive, tends to be significantly higher than that of ordinary workstations.

Our use of VEs for training in the electric power utility industry was introduced with the advent of ESOPE-VR, an operator training simulator (OTS) employing VE technology for cost-effective training of station personnel in the manual operation of switching equipment, at a level of realism that approaches real-world training but lacks the hazards associated with real-world operation [3, 4]. While ESOPE-VR has been enthusiastically received, the high cost associated with today's VEs is an obstacle to future development and commercialisation. A solution is presented here to recreate the VE on low-cost personal computers (PCs), reducing the cost of the VR software itself, yet preserving realism and real-time performance.

In this paper we present Mirage, a low-cost, PC-based VR development system, within the framework of ESOPE-VR. Through Mirage, development of PC-ESOPE-VR can provide the same or improved functionality as the original, ensuring support for a 3-D visual interface, voice recognition and feedback, navigation and manipulation facilities, and expert system, multimedia and multi-user (distributed) support, at a fraction of the cost. Moreover, PC-ESOPE-VR is compatible with ESOPE-VR by design. It is capable of drawing on existing ESOPE-VR power component libraries and, most notably, supporting the automatic synthesis of a virtual power station environment from its single-line diagram representation, an innovative method of virtual world generation unique to ESOPE-VR. The approach used via Mirage towards fulfilling all these issues is described, followed by key implementation results.

II. Background

A cursory introduction to ESOPE-VR and its impor-

0-7803-3713-1/97 $10.00 © 1997 IEEE

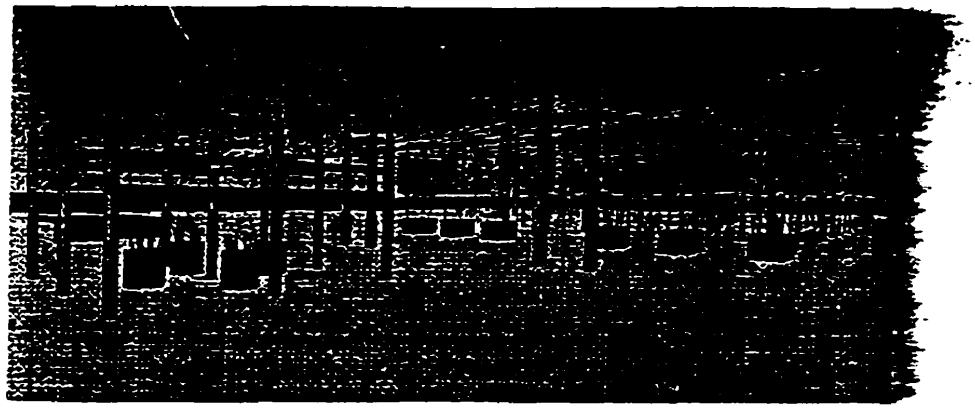


Figure 1: A view of an ESOPE-VR power station.

tance is necessary here to successfully present the role Mirage plays towards the development of PC-ESOPE-VR. Further elaboration of ESOPE-VR, and of ESOPE-VR's predecessor ESOPE, however, have been previously presented (in [3, 4]).

A. ESOPE and ESOPE-VR

Studies of VEs have demonstrated their value in the industry in design, manufacturing, maintenance and repair [5], with examples seen at NASA [2], within NATO [6] and in the US Navy [7]. By providing a level of realism in training that approaches real-world training but lacks the hazards associated with real-world operation, the effects of dangerous or irreparable operator error can be avoided, and risks to human or equipment safety removed. Non-VE OTS experiences within the power domain alone reflect a diminished "fear of error" and increased operator interest and effectiveness ascribable to familiarity with, and confidence in, the task at hand [8, 9].

ESOPE-VR is an OTS developed for Hydro-Québec consisting of a VE for the training of power station personnel in the manual operation of station switching equipment. From the operational perspective, the VE of ESOPE-VR includes: -

- 3-D graphical modelling of the power station environment with stereoscopy;
- sound immersion for added realism of the power station;
- speech recognition for verbal command input;
- interactive multimedia presenting photo and video sequences of actual power station environment and equipment behaviour for educational reinforcement; and
- interactive help and feedback via digitised voice messages.

Fig. 1 shows a view of an ESOPE-VR power station which was derived from the ESOPE training module single-line diagram and rendered using WorldToolKit. The scene contains approximately 8000 polygons. Fig. 2 shows a partial view inside the ESOPE-VR power station operator control room. Underlying ESOPE-VR are:

- an interface to ESOPE (a Windows-based operator training system known commercially as *Operator*→), and
- a software module for the automated synthesis of a 3-D power substation VE from any input custom single-line diagram representation.

ESOPE is a software package that consists of an expert system infrastructure and graphical user interface (GUI) for the procedural training of switching operations performed upon power station single-line diagrams. Fig. 3 shows a sample single-line diagram of an ESOPE training environment. ESOPE runs concurrently with ESOPE-VR to validate switching operations, and update state variables when changes are made to the network topology.

However, to support the VE immersiveness of ESOPE-VR requires two Silicon Graphics (SGI) Indigo workstations—one SGI R4000 for the rendering of the 3-D objects in Sense8's WorldToolKit, and one SGI R3000 for audio and video I/O. Two dedicated IBM-PCs are also required, one to accept verbal command input and the other to run the ESOPE simulator. Such resource requirements are nonetheless in keeping with those of typ-

ical VEs seen today, if not less lavish! Nor is the expense associated with VR and VE technology limited to hardware—software costs and licenses boost VE costs further. And so, while VEs for training show positive effects on job performance, with audio and visual cues acting as positive aids for trainees using the VEs, they are plagued by high cost. Faced with this daunting view, Mirage was developed. While Mirage does not aim to supplant all VR software and promote PC equipment exclusively, it demonstrates the feasibility of reducing costs by a factor of approximately ten.

B. Head-Coupled VR Display

The VR immersiveness actually used by ESOPE-VR is *head-coupled VR display*, popularly known as "fish tank", or "desktop", VR. It uses a regular workstation display to provide a less-ambitious virtual world in which the immersive effect is limited to the volume of space roughly equivalent to the inside of the display monitor. This is cost-effective in comparison to full immersion via head-mounted displays (HMDs) and spatially-immersive displays (SIDs). [10]

C. A Survey of VR Development Toolkits

While a comprehensive coverage of existing VR development software currently on the market or available for public use cannot be performed here, a brief survey of popular VR toolkits, or software function libraries for VR application development, is now presented to situate Mirage within the context of existing VR software. It will be seen in comparison that Mirage is capable of rivalling costly toolkits in general functionality that includes multi-platform portability.

OpenGL was developed by SGI to facilitate 3-D programming on SGI workstations. Having become a graphics language standard for VR software development on such high-end graphics processors, it paved the way for other VR development toolkits such as WorldToolKit, Minimal Reality Toolkit (MR Toolkit) and Open Inventor [11]. Support for OpenGL has now been extended to various UNIX systems including AIX, Solaris and PC systems such as OS/2, Windows 95 and NT. OpenGL's primary advantages lie in freeing the programmer from low-level 3-D programming, and, more significantly, in liberating the processor from many potentially-complex software computations by performing the OpenGL functions on dedicated graphics hardware, resulting in considerable performance improvement.

Sense8 Corporation's WorldToolKit remains popular despite its significant cost due to its well-designed VR function library. A large number of platforms (Microsoft Windows on PCs, and SGI workstations) as well as peripheral devices (the Logitech Cyberman, and the CrystalEyes LCD Shutter Glasses, to name a couple) are supported. As aforementioned, ESOPE-VR was originally implemented in WorldToolKit. A major problem with WorldToolKit is the licensing required for its use and, at the time of ESOPE-VR's development, the lack of support for multiple users within the virtual world. By contrast, MR Toolkit was developed at the University of Alberta and is freely available to educational institutions. Also developed on SGIs, it exploits UNIX networking capabilities to distribute processing requirements across networked processors, greatly enhancing application performance.

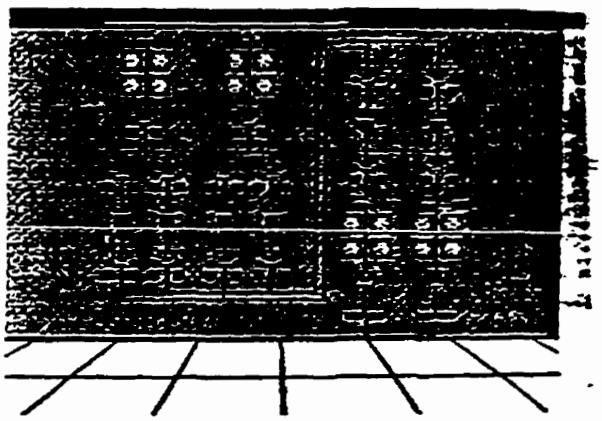


Figure 2: A partial view of the panel inside the ESOPE-VR operator control room.

Among the freeware VR toolkits available are Rend386, Avril and VR386 [12]. Rend386 is the collaborative work of Bernie Roehl and Dave Stampe of the University of Waterloo that initiated much of the general public to VR due to its availability on the Internet and runnability on low-end 386 PCs. This was made possible as a result of its design emphasis on fast rendering. Both VR356 and Avril evolved from Rend386. VR386 focusses on speed and performance through modular separation of assembler and C code to aid in the development of VR applications. Though Avril provides a C programming environment, its source code is not available. VR386 executes surprisingly well on low-cost PCs, but runs on MS-DOS and is limited to the 640 kB ceiling of conventional memory or, through some complex manipulations, extended memory at best. Clearly, under such memory limitations it would be impossible to develop sophisticated VEs. Although retaining the spirit of VR386 and much of its design philosophy, Mirage addresses the need for processing memory, porta-

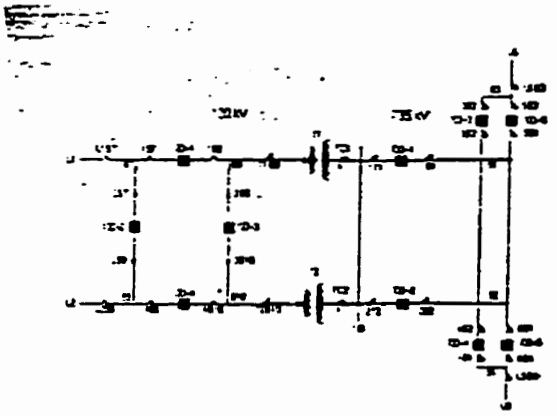


Figure 3: A typical single-line diagram displayed in ESOPE.

Figure 4: An ESOPE-VR power station, originally in WorldToolKit .NFF format, converted and rendered here in Mirage.

bility and multiprocessor networking. Mirage began as a C translation of VR386 for UNIX platforms running the X Windows protocol, including Sun and SGI workstation and Linux on PCs, but contains additional functionality such as texture mapping and 3-D file format conversion.

III. Mirage

Mirage is a virtual reality toolkit inspired from VR386, that runs on UNIX platforms in X Windows. The rendering of complex VEs is therefore freed from the limitations of MS-DOS conventional memory that is present in VR386. The underlying philosophy of Mirage is speed. It is with this purpose in mind that most Mirage design principles were adopted, some of which are explained here. Rendering of the ESOPE power station by Mirage is successfully performed in high resolution Super VGA on Pentium PCs, in support of low-cost VR.

Mirage possesses the capability of directly importing WorldToolKit .NFF files, such as the power station from ESOPE-VR in Fig. 4, complex 3-D objects created with AutoCAD through the DXF format, and the VR386 .PLG file format. The platforms currently supported are the SGI Indigo and Sun workstations, as well as PCs running Linux.

A. User Interface

Mirage uses the "fish tank VR" display principle described earlier to present a perspective view of the 3-D environment. Several virtual cameras may be placed in the world that enable the user to switch between these views using the function keys, the equivalent of teleporting to different locations. Navigation is presently achieved using the arrow keys on the keyboard, or by mouse clicks in one of four areas at the edges of the window, although other devices such as the PowerGlove and the joystick will be supported in the future.

B. Drawing and Rendering

The geometric primitives inherent to Mirage are points, lines and polygons; more complex geometric primitives must be tesselated to polygons by the user through software. Up to 9000 polygons may be displayed on screen at once but this limit can be extended as more powerful processors become available. Text may be displayed either by drawing polygons to form the text or by texture-mapping a text image onto a polygon. For example, identification numbers may be texture-mapped onto equipment for accurate identification. The PC-paintbrush, or .PCX, image format, to which .BMP, .TIF, .GIF and .JPG formats can easily be converted, is currently supported for texture-mapping. Also, polygons are flat-shaded. This type of shading is more rapid than other more realistic but complex shadings. Shaded polygons may have up to 16 different colors, with 16 intensities depending on the location and direction of the light source. The light source consists of an ambient light supplemented by either a point light or directional light source.

The rendering pipeline has the following features:

1. Objects not in the view volume are eliminated to prevent unnecessary processing.

2. Backface culling is performed, in which polygons not facing the viewer are removed, also to avoid processing.

3. Polygons extending outside the view volume are clipped.

4. In the last step, the polygons are sorted by their Z-depths, or their distances from the viewer. They are then drawn into the image buffer using the painter's algorithm. The triple buffering technique is used for smooth animation—the view is drawn in one of three memory buffers, all X Windows pixmaps, before being copied to the screen window.

Arithmetic operations are done both in floating- and fixed-point math. Fixed-point math allows quicker operations, as integer arithmetic is less CPU-intensive than floating-point math. Object positions and rotation angles are therefore specified in fixed-point 32-bit format.

C. Interaction and Animation

As in typical interactive VR programs, the Mirage program structure consists of one major loop that polls for user input events, modifies the database of virtual world objects, and refreshes the display accordingly. Fig. 5 illustrates this loop in flowchart format.

The VE's attributes and behaviour can be described in a "world" file of extension .WLD. This file contains the list of objects in the virtual world, their positions and orientations as well as other parameters such as the colour of the sky in the environment and the position of the light source. Also specifiable in this file are scripts which may be used to describe interactions and scene animations. These interactions with the user or animations on the part of the VE are, in fact, state machines that can be affected by the position of the user in the virtual world or by the selection of an object on the screen with, say, a mouse. Scene animation and interaction may be also be pre-programmed in C via Mirage library functions. The mechanised opening of a circuit breaker is an example of scene animation.

The .WLD file refers to objects that are described in another type of file with the extension .PLG. These 3-D objects may possess multiple levels of detail (LOD) dependent on their current size on the screen as a result of distance from view, to speed up the rendering process. For example, a transformer viewed from afar can be approximated by a simple box.

D. ESOPE-VR Conversion of Single-Line Diagrams

From the ESOPE single-line diagram, such as in Fig. 3, the interpreter creates its three-dimensional representation. The topology of the circuit as well as the positions of the equipment in the single-line diagram determine this representation. The translation procedures are summarised as follows: The circuit topology is analysed, resulting in a simplified representation of the circuit. Then,

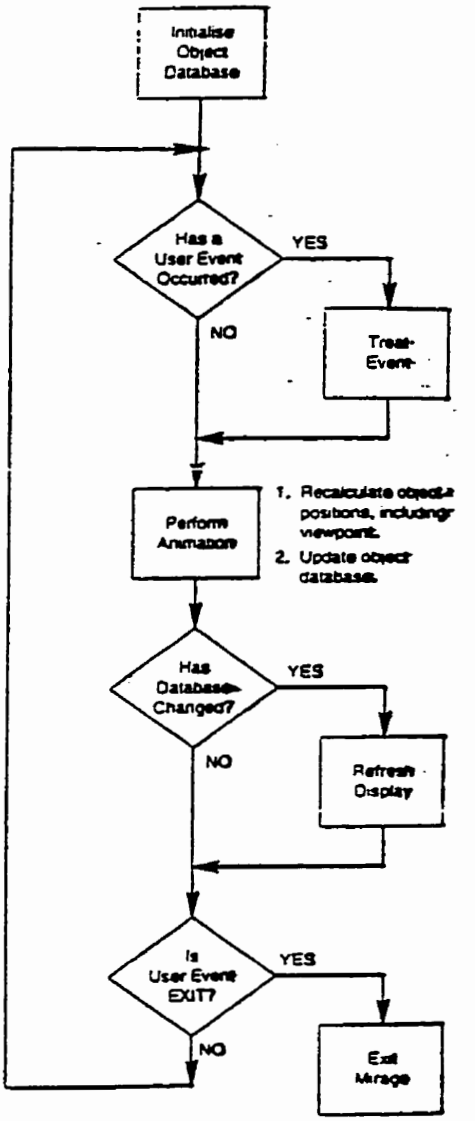


Figure 5: A flowchart representation of Mirage program structure.

interpreter matches this representation with one of several possible station bus schemes [13].

a. single bus.

b. double bus, double breaker,

c. main and transfer bus,

d. double bus, single breaker,

e. ring bus.

f. breaker and a half or inverted breaker and a half, or

g. breaker and a third.

A number of quantities are evaluated, such as the number of breakers that must be opened to isolate a line, which together form criteria for determining the corresponding "ideal" configuration. The 3-D representation can then be created. The interpreter relies on a library of equipment models to place equipment in its position and generates the equipment interconnections. This library contains equipment dimensions, spacing, ratings, as well as connection position coordinates. The station is created in a .WLD file, in the form of a list of objects from the equipment library with their coordinates, orientation and corresponding ID number. Mirage reads this file, and displays the station on-screen.

E. Performance

We have found that the 60-MHz Pentium computer has a performance comparable to the SGI R4000 which costs approximately ten times more. Table 1 shows the performance of Mirage in rendering a typical ESOPE-VR power station containing 8175 polygons in 8-bit colour mode in a 640-by-480 pixel window. The performance of the Sun Sparc-5 may be less than expected due to network delays.

Table 1: Mirage performance results.

Computer Platform	Loading Time	Frames/s
SGI Indigo R4000	20.5s	2.3
SGI Indigo R3000	40s	1.25
Pentium 60 MHz	25.5s	1.9
Sun Sparc-5	64.5s	1.5

IV. Discussion and Future Work

UNIX workstations running the X Window interface have become the basic system building blocks in SCADA and energy management systems (EMSs). Personal computers (PCs), on the other hand, due to the current perception of unreliability, replace the workstations only at the lower end of the marketplace [14, 15]. The acceptance of the PC in the power utility domain has been aided by the popularity of Microsoft Windows and associated commercial software. To support the power industry's use of X-workstations and terminals, Mirage has been implemented in X Windows which is supported on many platforms including the PC. Through this portability, Mirage, and thereby PC-ESOPE-VR, can be extended to communicate with other existing control and data acquisition software across a variety of platforms to create realistic VE-OTS scenarios based on real data input, to import OTS configuration data, or to export trainee performance statistics directly to reports or spreadsheets.

ESOPE is primarily used by personnel to determine the topology of power station networks, to locate equipment and to evaluate the effects of switching operations [4]. The PC-ESOPE-VR environment as rendered by Mirage proves the feasibility of building low-cost VEs on Intel processor-based PC technology for use in operator training.

We are currently developing a VR interface to WITS [16], a CLIPS-based expert system developed for the training of alumino-thermal welders at Hydro-Quebec using Mirage. The WITS expert system was designed for reusability to accomodate both incremental and monumental updates to training procedure, all achievable through straightforward changes to the WITS database. Therefore, a Mirage-WITS VE-OTS design is customisable at different levels of granularity to suit various user training requirements. We are also investigating multi-user VR in Mirage, making it possible for several users to interact inside the same VE with one another.

Mirage can be freely obtained by anonymous FTP following the instructions at our Web sites:

http://www.gegi.polymtl.ca/electech/marceau/index.html
http://www.vr.ee.mcgill.ca

V. Acknowledgments

The authors thank NSERC, FCAR and Le Fonds de Polytechnique for their financial assistance. Their thanks extend also to the members of the original ESOPE-VR team: Etienne Garant, Alex Okapuu-von Veh, Amir Shaikh, Alain Daigle, Louis Marquis and Robert Gauthier.

References

[1] O. Hagsand, "Interactive Multiuser VEs in the DIVE System", *IEEE Multimedia*, Vol. 3, No. 1, Spring 1996, pp. 30–39.

[2] R. B. Loftin, P. J. Kenney, "Training the Hubble Space Telescope Flight Team", *IEEE Computer Graphics and Applications*, Vol. 15, No. 5, Sep. 1995, pp. 31–37.

[3] A. Okapuu-von Veh, R. J. Marceau, *et al.*, "Design and Operation of a Virtual Reality Operator-Training System". Paper No. 96WM 157-3 PWRS. *IEEE Transactions on Power Systems*, Vol. 11, No. 3, Aug. 1996, pp. 1585–1591.

[4] E. Garant, P. Desbiens, *et al.*, "Three-Dimensional Modelling for a Virtual Reality Operator Training Simulator". *Procs., Stockholm PowerTech Conference*, Stockholm, Jun. 18–22, 1995, pp. 31–36.

[5] A. K. Noor and S. R. Ellis, "Engineering in a virtual environment". *Aerospace America*, Vol. 34, No. 7, Jul. 1996, pp. 32–37.

[6] S. Stansfield, N. Miner, *et al.*, "An Application of Shared Virtual Reality to Situational Training". *Procs., IEEE Virtual Reality International Symposium (VRAIS'95)*, pp. 156–161.

[7] D. Zeltzer, N. J. Pioch and W. A. Aviles, "Training the Officer of the Deck", *IEEE Computer Graphics and Applications*, Vol. 15, No. 6; Nov. 1995, pp. 6–9.

[8] S. Rajagopal, M. Rafian-Naini, *et al.*, "Workstation-Based Advanced Operator Training Simulator for Consolidated-Edison", *IEEE Transactions on Power Systems*, Vol. 9, No. 4, Nov. 1994, pp. 1980–1986.

[9] G. Miller, A. Storey, *et al.*, "Experiences Using the Dispatcher Training Simulator as a Training Tool". *IEEE Transactions on Power Systems*, Vol. 8, No. 3, Aug. 1993, pp. 1126–1132.

[10] K. W. Arthur, K. S. Booth and C. Ware, "Evaluating 3D Task Performance for Fish Tank Virtual Worlds", *ACM Transactions on Information Systems*, Vol. 11, No. 3, Jul. 1993, pp. 239–265.

[11] R. S. Kalawsky, *The Science of Virtual Reality and Virtual Environments*, Addison-Wesley, Workingham, England, 1993.

[12] J. Gradecki, *The Virtual Reality Programmer's Kit*, John Wiley & Sons, Inc., New York, New York, 1994.

[13] D. G. Fink, H. W. Beaty, (eds.), *Standard Handbook for Electrical Engineers*, 11th Ed., McGraw-Hill, New York, New York, 1978, p. 17-3.

[14] P. Sigari, M. Rafian-Naini and K. J. Sluis, "Portable and Affordable Operator Training Simulators", *IEEE Computer Applications in Power*, Jul. 1993, pp. 39–44.

[15] W. E. Chainey and W. R. Block, "Recent Advances in Master Station Architecture", *IEEE Computer Applications in Power*, Apr. 1994, pp. 24–29.

[16] E. K. Tam, P. Allard, *et al.*, "WITS: A Reusable Architecture for a VR-Based ITS". *Intelligent Tutoring Systems (ITS'96) Workshop for Architectures and Methods for Designing Cost-Effective and Reusable ITSs*, Montreal, P.Q., Jun. 12–14, 1996.

VI. BIOGRAPHY

Elaine K. Tam completed her B.Eng (Computer) at McGill University in 1993. Since then she has worked on custom high-speed data acquisition systems at MPB Technologies, Inc., in Montreal, and is now pursuing a Master's degree at McGill. Her immediate research interests include computer graphics and animation, and intelligent tutoring systems in virtual environments.

Christophe Maurel obtained his B.Eng. at École Polytechnique de Montréal (1994) where he is currently a Masters student. His research interests include computer graphics and virtual reality operator training systems.

Patrick Desbiens obtained his B.Eng. at École Polytechnique de Montréal (1994) where he is currently a Masters student. His research interests include power systems analysis and virtual reality operator training systems.

Richard J. Marceau obtained his B.Eng. at McGill University (1977), his M.Sc.A. at École Polytechnique de Montréal (1983) and his Ph.D. at McGill University (1993). From 1978 to 1982 he worked for MONENCO in system planning, relaying and power station design. At Hydro-Québec from 1982 to 1990, he first worked as operations engineer, later as a researcher in induction heating (IREQ), and in strategic planning of R&D. He is currently Assistant Professor at École Polytechnique. His research interests include dynamic security analysis, artificial neural network technology for power systems and virtual reality operator training systems.

Alfred S. Malowany obtained his B.Eng. (1959), M.Eng. (1962) and Ph.D. (1967) from McGill University where he is currently an Associate Professor in Electrical Engineering. His research interests include Robotics, Medical Informatics, Real-Time and Virtual Reality Systems.

Louis Granger obtained a B.Sc in Mathematics in 1967 and a M.Sc. in 1969 from Université de Montréal. He is currently an associate professor of Electrical and Computer Engineering at École Polytechnique de Montréal. His research interests are computer graphics, simulation and virtual reality systems.